Learning Private CNN Language Models

Mila Schultz
Department of Computer Science
Stanford University
Stanford, CA 94305
milafaye@stanford.edu

Abstract

The combination of deep learning and large corpuses of text has enabled many powerful language models to be trained. However, privacy concerns may make training models directly on all text at a central server undesirable or even impossible. To maintain user privacy, a central model which can be trained on user data while keeping such data private and local to the user would be ideal. We have built an efficient convolutional neural network (CNN) language model which has been modified to use federated learning and differential privacy during training. The performance of the federated, private model is comparable to the public, central model.

Mentor: Michael Hahn

1 Introduction

Privacy leaks are commonplace in today's connected social media landscape. Consumers have concerns about social media and other companies' use of their personal data. Further, directed attacks on security of user information have been carried out. These factors have lead to increased interest in training models while guaranteeing user privacy. Even trained models are vulnerable to leaking private data and have been shown to memorize private data. In light of these factors, users may choose to keep their data private, or platforms may prevent access to user-level data. To maintain user privacy, a central model which can be trained on user data while keeping such data private and local to the user would be ideal.

Language models have been trained for many purposes in natural language processing, including machine translation. These models depend on the availability of a large, representative corpus of text, and, in particular, a model that learns the structure and is predictive of text-based personal user data requires training with vast amounts of sensitive text data.

In the past, RNN and LSTM language models have been state of the art. More recently, transformer and CNN architectures have been of interest for performance and efficiency. The computations performed by CNN architecture lend well to to parallelization, and since CNN models are commonly used in vision tasks, mobile devices have optimized GPUs and even specialized hardware to efficiently train and run CNN models.

We build and train a language model that federates training and ensures user privacy. We compare the performance of this model with non-private and centrally trained equivalents. The private, federated language model performs comparably to non-private federated equivalents.

1.1 Motivation

2 Related Work

2.1 CNNs for NLP

CNN models have achieved excellent performance in image and vision tasks over the last decade. CNN models include convolution layers which extract features from the data and are easily parallelized on a GPU due to their local repeated computations. The efficiency of CNN models has led to interest in training natural language tasks on such architectures. Kim [2014] achieved excellent results on sentence classification benchmarks using a CNN. A character-level CNN model by Conneau et al. [2016] achieved state of the art on classification tasks. Their model is comparable to a deep CNN traditionally used in vision and includes convolutional blocks with shortcuts that allow input to bypass convolutions. Shortcuts or gating improve performance and are key elements of CNN language models.

Dauphin et al. [2016] introduce a CNN language model which provides the basis for the CNN language model in this work. Their model achieved competitive results on the WikiText-103 and Google Billion Words benchmarks. The architecture involves stacking gated convolutional blocks, which benefit from parallel computation. The architecture will be discussed further in Section 3.1.1.

2.2 Federated and Private Learning

In an ideal world, a trained model would learn general patterns in the data without memorizing or reproducing any particular training example. Protecting individual privacy during training can be accomplished both by federating the training and modifying the training algorithm to restrict the amount of information stored in the model from each individual training example.

Federation, or federated learning, means that the model is sent to multiple clients to be trained rather than being trained on a central server. The training algorithm `FederatedAveraging` was introduced by McMahan et al. [2016] as an efficient protocol for federating learning between multiple clients and computing a weighted average of model updates.

The goal of training a model privately is to ensure that users' data is not compromised by access to the model. Dwork [2011] introduced differential privacy, which is a formalization of a privacy guarantee. The framework of differential privacy provides a formal privacy guarantee about how much information from the user training data can be recovered from the parameters of the trained neural network. Differential privacy is defined in McMahan et al. [2018] as:

Def: Differential Privacy A randomized mechanism $M : D \rightarrow R$ with a domain D (possible training datasets) and range R (all possible trained models) satisfies (ϵ, δ)-differential privacy if for any two adjacent datasets $d, d' \in D$ and for any subset of outputs $S \subseteq R$ it holds that $\Pr[\mathcal{M}(d) \in \mathcal{S}] \leq e^{\epsilon} \Pr[\mathcal{M}(d') \in \mathcal{S}] + \delta$.

In the case of training a model, the training procedure is the mechanism M, and an output is a trained model. McMahan et al. [2018] defines two datasets d and d' to be **user-adjacent** if one of the datasets can be formed from another by adding the data from one user. The privacy guarantee is a bound on the probability that a particular user's data was included in the training of the model. Equivalently, given two results of a query to the user data resulting in the model, there must be a way to bound the difference between the models. In this case, we will bound the queries and add noise proportional to the bound. In their work, McMahan et al. [2018] train a differentially private recurrent neural network (RNN) language model using long short-term memory (LSTM) and find that their private language models have similar performance to non-private language models but require more computation to achieve the equivalent performance.

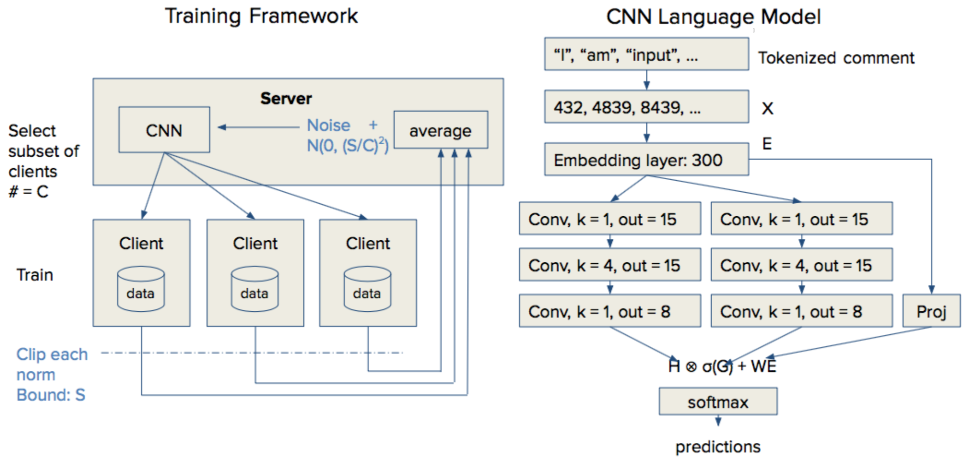

Figure 1: Training framework and model architecture.

3 Approach

3.1 Architecture

3.1.1 CNN Language Model

The CNN language model architecture can be seen in Figure 1. The model takes as input a sequence of words $\mathbf{X} = x_0, ..., x_N$ of length N. The embedding $\mathbf{E}$ for the input is looked up with an embedding layer. The embedding is then passed through a gated convolution block hidden layer. Let m be the number of input channels and n be the number of output channels to the gated convolution block hidden layer. Let $Z \in \mathbb{R}^{N \times m}$ be the input to the hidden layer. Let k be the kernel size, and h be the hidden layer bottleneck size. The hidden layer applies three convolutions to the input:

$$H_1 = (Z * W_1 + b_1) \tag{1}$$

$$H_2 = (H_1 * W_2 + b_1) \tag{2}$$

$$H_3 = (H_2 * W_3 + b_3) \tag{3}$$

where $W_1 \in \mathbb{R}^{1 \times m \times h}$ and $b_1 \in \mathbb{R}^h$, $W_1 \in \mathbb{R}^{1 \times h \times h}$ and $b_2 \in \mathbb{R}^h$, $W_3 \in \mathbb{R}^{1 \times h \times n}$ and $b_3 \in \mathbb{R}^n$. In parallel, the hidden layer applies the same operations with different trained weights to the input:

$$G_1 = (Z * V_1 + c_1) \tag{4}$$

$$G_2 = (G_1 * V_2 + c_1) \tag{5}$$

$$G_3 = (G_2 * V_3 + c_3) \tag{6}$$

where $V_1 \in \mathbb{R}^{1 \times m \times h}$ and $c_1 \in \mathbb{R}^h$, $V_1 \in \mathbb{R}^{1 \times h \times h}$ and $c_2 \in \mathbb{R}^h$, $V_3 \in \mathbb{R}^{1 \times h \times n}$ and $c_3 \in \mathbb{R}^n$. The output of the block is:

$$H_l = Z + H_3 \otimes \sigma(G_3) \tag{7}$$

The function of $\sigma(G_3)$ is to act as a gate for the output, and there is a residual connection.

The input to the convolution is left zero-padded with $k - 1$ elements to prevent the sequence from cheating at the task. The layer includes a bottleneck for efficiency, as the convolution is only applied with the full kernel size in the middle layer of the gated convolution block.

If the input and output layers of the block are not the same size, that is if $m \neq n$, then a projection without bias is applied to the residual Z to ensure that the dimensions align.

The CNN model was implemented in PyTorch with reference from and modification of Bressler [2018], who implemented the model from Dauphin et al. [2016]. Some code to process vocabulary files from Stanford University CS224N assignments was repurposed.

3.1.2 Federation Framework

A framework to federate training was built with the ability to send models, parameters, and parameter deltas between a simulated client and server. The framework implements the training algorithms described in Section 3.2 and is implemented with Python and PyTorch.

3.2 Algorithms

3.2.1 Federated Averaging

The training process for the CNN model has been modified to use federated learning. Specifically, the algorithm `FederatedAveraging`, also known as `FedAvg`, as introduced by McMahan et al. [2016] is used. In this algorithm, the training occurs over multiple rounds. During each round, the server sends its model to a random set of clients of size C. Each client performs training on the model using its data. The clients send the models back to the server. The server then updates its model parameters to be an average of each client's model parameters, weighted by the proportion of examples each model used for training during the round.

3.2.2 Private Federated Model

Modifications are made to the `FedAvg` algorithm to achieve additional user-level privacy. The algorithm `DP-FedAvg` was described by McMahan et al. [2018] includes four key modifications:

1. Select clients for training independently, leading to randomly sized batches per round
2. Clip each client's update to the model by L2 norm
3. Average the updates in a way that provides privacy
4. Use Gaussian noise in the final central server's update at the end of each round

The intuition behind these modifications is that the server will add noise proportional to the norm of each client's update in order to bound the contribution of each client to the model.

For simplicity, we assume that each client has an equal number of training examples. The result is that the model updates from each client should be equally weighted by the server. We have K total clients, and let $\tilde{C}$ be the expected number of clients per round. We choose $\tilde{C} = 100$ for our experiments. During each round, each client is selected for training during that round with probability $q = \frac{\tilde{C}}{K}$.

The training algorithm, a version of `DP-FedAvg` with hyperparameters chosen to simplify the calculations, is shown in Algorithm 1. During each round of training, the server makes a query $f(C_{train})$ to a subset size $\tilde{C}$ of clients. The clients each train the model, thereby leaking user information into each model update. The aggregation of these $\tilde{C}$ model updates are the result of the server's query. Each of the model updates is bounded by S with clipping. To ensure privacy in the final model, the server adds Gaussian noise proportional to S, with $\sigma = \dfrac{S}{\tilde{C}}$. The noise ensures that the sensitivity is bounded:

$$\|f(C_{train}) - f(C_{train} \cup \{k\})\| < S \tag{8}$$

where k is any other client not in C_{train}. McMahan et al. [2018] show that with appropriate hyperparameters, `DP-FedAvg` achieves differential privacy. The parameters (ϵ, δ) are calculated with the TensorFlow Privacy library [Google, 2019], as discussed in Section 5.

4 Experiments

4.1 Data

We train the models on a dataset of comments posted on the public social media website Reddit during December 2018 [Reddit, 2019]. The dataset is hosted by Google BigQuery and was processed to aggregate comments by the same author into author-specific subsets of the data. Each subset by a single author is a simulated user-level dataset in the training algorithm, termed client dataset. The client datasets are lower-cased, tokenized, and clipped to 4000 word tokens each. Datasets with fewer than 4000 tokens are not used. We use 22,389 client datasets for a total of 89,556,000 word

Algorithm 1 `DP-FedAvg Simplified`

```
Server Training:
    for each round t = 1, ... do
        $C_{train} \leftarrow$ (sample users with probability $q = \frac{\tilde{C}}{K}$)
        for each user $k \in C_{train}, \dots$ do
            $\Delta_k^{t+1} \leftarrow$ ClientUpdate($k$, $\theta^t$)
        $\Delta^{t+1} \leftarrow \frac{\sum_{k \in C_{train}} \Delta_k^{t+1}}{\tilde{C}}$
        $\sigma \leftarrow \frac{S}{\tilde{C}}$
        $\theta^{t+1} \leftarrow \theta^t + \Delta^{t+1} + \mathcal{N}(0, \sigma^2)$
ClientUpdate($k$, $\theta^0$):
    $\theta \rightarrow \theta^0$
    for each local epoch do
        for batch $b$ in batches do
            $\theta \leftarrow \theta - \alpha \nabla \ell(\theta; b)$
            $\theta \leftarrow \theta^0 +$ Clip($\theta - \theta^0$)
    return update $\Delta_k = \theta - \theta^0$
Clip($\Delta$):
    return $\Delta \cdot \min(1, \frac{S}{\|\Delta\|})$
```

tokens. We have assumed that the majority of Reddit comments are written in English, although some proportion of our dataset may be non-English.

The word embeddings are trained as part of the model, but since the training dataset is private to the server, the vocabulary for the embeddings must be computed without access to the training data. We use the vocabulary (but not the trained weights) of the top 10,000 words from GloVe pre-trained word vectors which have been trained on Wikipedia 2014 and Gigaword 5 datasets [Pennington et al., 2014].

4.2 Evaluation

The models are evaluated on a language model prediction task. The evaluation dataset is a separate held out dataset of Reddit comments containing 113,000 word tokens. The metrics used are cross entropy loss, perplexity, and top-1 accuracy of the prediction. Top-1 accuracy refers to the fraction of tokens which are accurately predicted by the model as having the highest probability.

4.3 Experimental Details

After hyperparameter tuning, models were trained using combinations of central or federated training, method of selecting clients to train per round, client update clipping, and server model update noising. A total of twelve representative models were trained.

For models trained with the `FedAvg` algorithm, a set of $C = 100$ randomly chosen clients participated in training each round. Models trained with the DP-FedAvg training model, an expected client set size of 100 used. Since each client is chosen to participate in training with probability q, client training set sizes vary per round. Hyperparameters used were a learning rate of 0.01, minibatch size of 10, sequence length of 8, convolution kernel size of 4. The embedding is dimension 300 with a vocabulary size of 10,000. The bottleneck layer in the gated convolution block is 15. There is one gated convolution block layer. The models trained for 20 rounds. Each client performed one epoch ($E = 1$) of training on its dataset if it participated in a round. During training, the models were evaluated on the held out test set after every communication round in the case of federated training or after the model trained each set of 100 client datasets in the case of central training.

Values of $S = 5$, $S = 10$, and $S = 20$ were used for the hyperparameter S, the bound for the L2 norm of the client update to the model. Both the L2 norm of the client update and the magnitude of the noise depend on S.

Table 1: Round during which best performance in perplexity, loss, and accuracy were achieved. FedAvg refers to federated training with a constant number of clients per round, DP-FedAvg refers to federated training with an expected number of clients per round, and Central refers to training a single model on the server.

Training	**Privacy**	**Perp**	**Round**	**Loss**	**Round**	**Accuracy**	**Round**
DP-FedAvg	Clipped, Noised, S=20	42.21	7	3.74	7	0.55	14
Central	Non-private	39.13	8	3.67	8	0.63	14
DP-FedAvg	Noised, S=20	34.7	5	3.55	5	0.56	8
DP-FedAvg	Non-private	25.94	20	3.26	20	0.59	18
DP-FedAvg	Clipped, S=5	24.56	20	3.2	20	0.59	20
DP-FedAvg	Clipped, Noised, S=5	23.6	20	3.16	20	0.60	20
DP-FedAvg	Noised, S=10	22.72	14	3.12	14	0.61	20
DP-FedAvg	Noised, S=5	21.12	20	3.05	20	0.60	18
DP-FedAvg	Clipped, Noised, S=10	20.13	20	3.0	20	0.62	17
FedAvg	Non-private	17.69	20	2.87	20	0.61	16
DP-FedAvg	Clipped, S=20	16.41	7	2.8	7	0.64	8
DP-FedAvg	Clipped, S=10	14.27	20	2.66	20	0.66	18

Table 2: Values of (ϵ, δ)-privacy by DP-FedAvg after 20 rounds of training.

ϵ	δ
1.44	10^{-4}
1.7	10^{-5}
1.95	10^{-6}
2.21	10^{-7}

4.4 Results

We evaluate the CNN language model performance when trained centrally. We compare the federated learning algorithms in terms of the training characteristics and model performance. Finally, we analyze the effect of the private training hyperparameters. In summary, all models perform reasonably on the task; the federated models outperform the centrally-trained model; and the private models can achieve equal performance to the non-private models.

The model training results are shown in Table 1. Accuracies of $0.55 - 0.66$ and perplexities of $14.27 - 42.21$ were achieved by the models, which is adequate performance. The CNN language model trained centrally achieved best perplexity of 39.13 on the test set at round 8. Notably, all but one of the models achieved lower perplexity. The only model which was unable to achieve lower perplexity was the DP-FedAvg model with clipping and noise with $S = 20$, the highest amount of noise added to the model.

As shown in McMahan et al. [2018], the algorithm DP-FedAvg can use the method to compute the privacy spent in training introduced by Abadi et al. [2016]. The differential privacy parameters (ϵ, δ) are calculated based upon the hyperparameters of training and shown in Table 2. Notably, the privacy indicated by ϵ is less than 10, indicating a only small loss of privacy for client-level data.

5 Analysis

The federated models achieved similar or superior performance to the unfederated, centrally-trained model. Figure 2 shows the test accuracy and loss of the training methods during training. The model trained centrally fit during the first training round, and the performance did not substantially improve during subsequent rounds, likely due to overfitting the training set. The model trained with FedAvg which does not add privacy modifications to the training achieves similar accuracy and better loss and perplexity than the central model, and fits substantially during the first few training rounds. The averaging of the client models can be understood as a form of regularization that improves the generalization of the centrally-trained model. The model trained with DP-FedAvg, using variable sized batches, but no privacy (no clipping or noise) is slower to approach the performance of the FedAvg model and performs slightly worse. After additional communication rounds, the perplexity

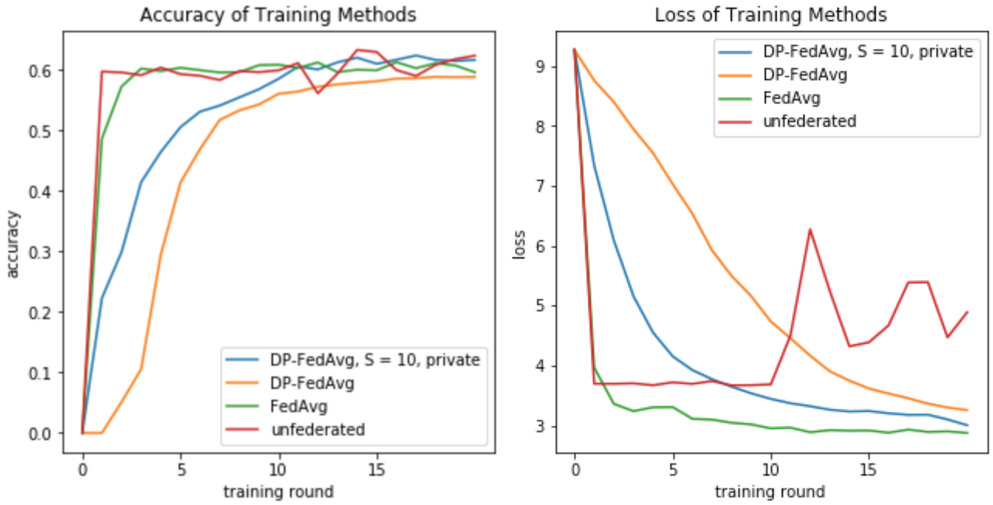

Figure 2: Accuracy and cross-entropy loss on the test set over 20 rounds of training. Note that perplexity = exp(loss).

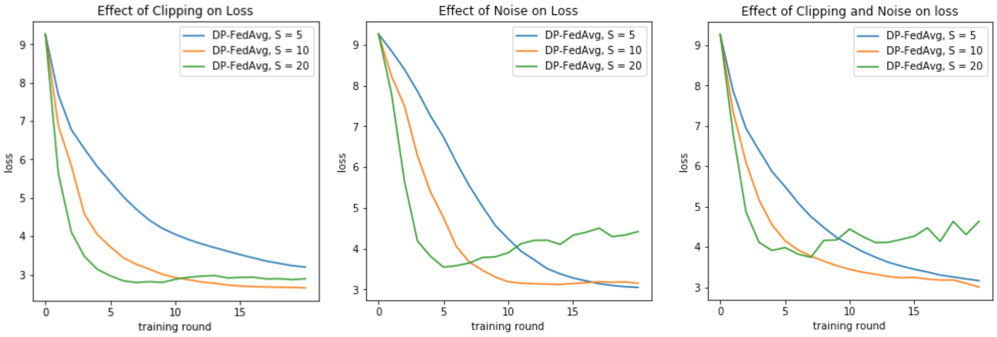

Figure 3: Cross-entropy loss on the test set over 20 rounds of training with varying bound S. Note that perplexity = exp(loss).

gap is 8.21 between the models may have closed further with extra training. The best-performing private DP-FedAvg model with clipping and noise outperformed the non-private DP-FedAvg and converged more quickly. The privacy modifications to the algorithm caused the training to be more well-behaved and may have acted as regularization as well.

Next, to analyze the impact of the privacy modification to DP-FedAvg, we investigate the behavior of DP-FedAvg models trained with and without each modification. We compare the models trained with client updates clipped using bounds $S = 5$, $S = 10$, and $S = 20$. The more aggressive the clipping, the slower the convergence; however, clipping with $S = 20$, the loosest bound, may have led to overfitting, as performance in both accuracy and perplexity decreased after round 10. See Figure 3.

We compare the effects of noise with bounds $S = 5$, $S = 10$, and $S = 20$. Recall that Gaussian noise is added with $\sigma = \frac{S}{C} = \frac{S}{100}$, leading to noise values of $\sigma = 0.05$, $\sigma = 0.1$, and $\sigma = 0.2$. The models trained with more noise converged and achieved best performance most quickly. The performance of the models trained with less noise converged more slowly, but achieved better performance eventually. The models trained with $S = 5$ and $S = 10$ achieved similar performance given that the $S = 10$ model had more training time. The noise has a regularization effect on training and aids in generalization, as seen in Figure 3.

Three models were trained with both client update clipping and Gaussian noise added to the server update with $S = 5$, $S = 10$, and $S = 20$. While the model with $S = 10$ converged the most quickly, the high amount of noise added to the updates cause the model to stop improving around round 5. The model with $S = 5$ converged more quickly than $S = 10$ in terms of perplexity but achieved similar performance, as seen in Figure 3.

We compare the effects of the clipping and noise combined compared with either clipping or noise. Using bound $S = 10$, the model with clipping only converged the most quickly and achieved the best performance. The model with noise only did not converge as quickly as the other two models early in training, but then achieved similar performance as the model with clipping and noise. The model with clipping and noise achieved similar performance in early rounds of training, but performance lagged slightly to a final perplexity gap of 5.86 with the model with clipping only.

The centrally trained CNN model did not include regularization or dropout in the architecture or hyperparameters. It is likely that some form of regularization would improve the performance of the centrally trained CNN language model, and in practice such regularization would be added to avoid overfitting on the training set. Each client achieves lower loss on the local model before the model averaging takes place on the server. We are encouraged that the modifications to the training algorithm to ensure privacy have the secondary effect of regularization. Models and training algorithms can be explicitly designed with the regularization of the privacy mechanism in mind.

6 Conclusion

The CNN language model successfully trains with central, federated, and privately federated training. The differentially private language model achieves superior performance to a naively centrally trained model. The private, federated language model performs comparably to non-private federated equivalents, and we see that modifications to training that enable privacy also act as regularization.

6.1 Future Work

In the future, we plan to train a larger model on a larger subset of the data, as the model trained was smaller than state-of-the-art. We plan to design a model with the privacy guarantee behaving as regularization and perform model optimization and hyperparameter searching with both the restrictions and benefits of differential privacy in mind. The

Acknowledgments

Thank you to Professor Manning and the CS 244N course staff for their support over the course. Thank you especially to Michael Hahn for his invaluable guidance while completing this project.

References

Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. URL `http://arxiv.org/abs/1408.5882`.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781, 2016. URL `http://arxiv.org/abs/1606.01781`.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. *CoRR*, abs/1612.08083, 2016. URL `http://arxiv.org/abs/1612.08083`.

H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016. URL `http://arxiv.org/abs/1602.05629`.

Cynthia Dwork. *Differential Privacy*, pages 338–340. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5_752. URL `https://doi.org/10.1007/978-1-4419-5906-5_752`.

H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJ0hF1Z0b`.

David W. Bressler. Pytorch implementation of dauphin et al. (2016) "language modeling with gated convolutional networks". `https://github.com/DavidWBressler/GCNN`, 2018.

Google. Tensorflow privacy. https://github.com/tensorflow/privacy, 2019.

Reddit. Reddit comments dataset. https://bigquery.cloud.google.com/dataset/fh-bigquery:reddit_comments, 2019.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 308–318, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4139-4. doi: 10.1145/2976749.2978318. URL http://doi.acm.org/10.1145/2976749.2978318.

7 Appendix

7.1 Figures

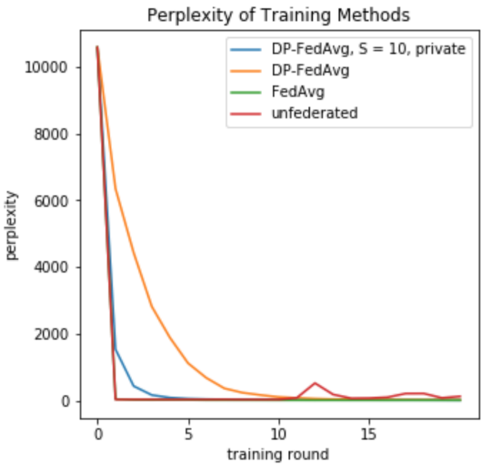

Figure 4: Perplexity on the test set over 20 rounds of training. The plot compares performance of different training algorithms.

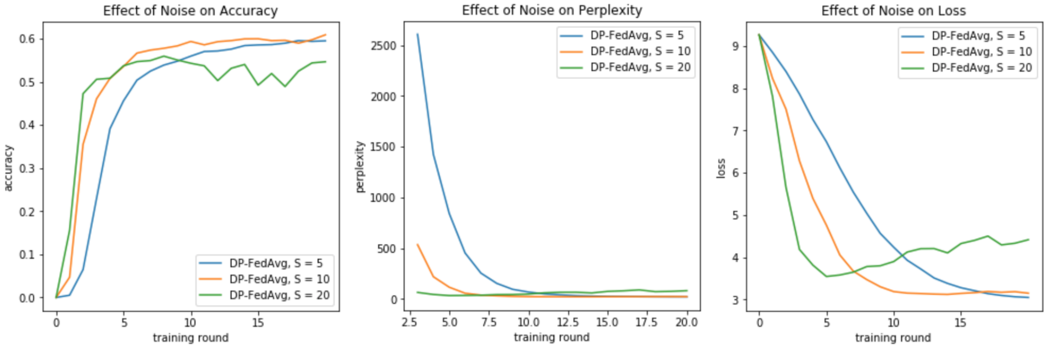

Figure 5: Accuracy, perplexity, and cross-entropy loss on the test set over 20 rounds of training. The plots compare performance of models with different amounts of noise.

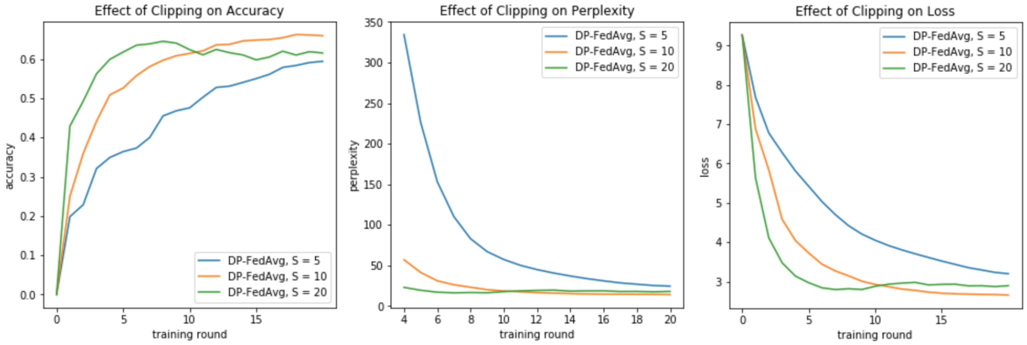

Figure 6: Accuracy, perplexity, and cross-entropy loss on the test set over 20 rounds of training. The plots compare performance of models with different clipping bounds.

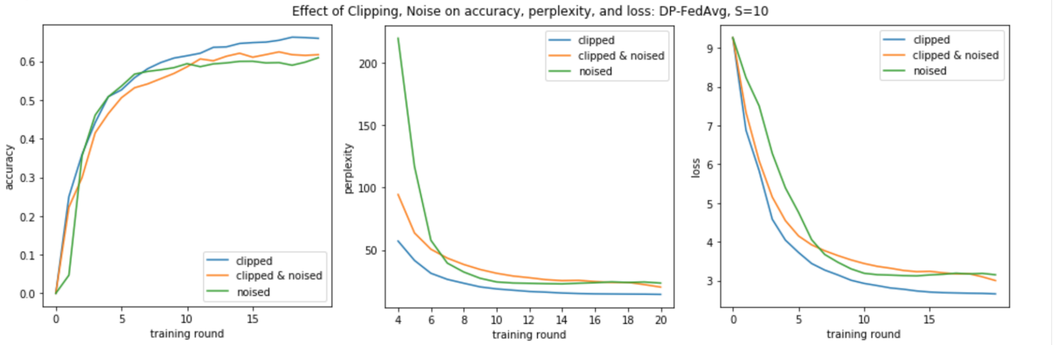

Figure 7: Accuracy, perplexity, and cross-entropy loss on the test set over 20 rounds of training. The plots compare different privacy modifications for bound $S = 10$.