# Components of Differential Private Stochastic Gradient Descent and its Effects on Memorization in Fine-Tuned Large Language Models

Leland Ling

CS599

Memorization in machine learning and deep learning models is an important subject as these models aim to capture underlying patterns that generalize well to unseen data. While memorization can lead to high performance on training and evaluation, a generative sequence model can unintentionally memorize phrases within the training set to such a high degree that it may spit out those phrases outright when prompted specifically. This phenomenon becomes an issue when these generative sequence models are trained on sensitive data. A motivating example would be Google's Smart Compose, a commercial text-completion neural network trained on millions of user email messages. Users of this tool may find that when prompted a sensitive phrase (such as "My social security number is ...") may return some kind of important information or specific details pertaining to someone. This phenomenon may be utilized by someone with ill-intentions to attack the model - entering specific details to hopefully glean some kind of important information. Therefore countermeasures against memorization that allow for generative sequence models to pick up on underlying patterns without outright learning phrases are needed.

## 1 Introduction

In this project, I aim to analyze Differential Privacy (DP), specifically components of Differential Private Stochastic Gradient Descent (DP-SGD), as a countermeasure to memorization in the context of fine-tuning. The components that I aim to analyze are clipping and random noise.

### Prior Works

I base this project mainly off the paper The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks written (Carlini et al. (2019)). The Secret Sharer paper uses a Long Short Term Memory (LSTM) model trained on the Penn Treebank (PTB) dataset. I opted to use a Gated Recurrent Unit (GRU) model taken from Tensorflow (Abadi et al. (2015)) and trained on the Shakespeare dataset from LEAF (Caldas et al. (2019)). LEAF's Shakespeare dataset is split into different shards associated with different characters (e.g. King Lear and his spoken phrases within the play The Tragedy of King Lear). Each of these shards are split into into training and validation sets prior to use. I also followed the Learning Differentially Private Recurrent Language Models paper from McMahan et al. (2018) when training and

fine-tuning.

In order to understand memorization, the Secret Sharer paper trains their model on the previously mentioned PTB dataset augmented with a number of "canary" phrases. They evaluate loss and their proposed log perplexity metric to measure how "surprised" the model is to see a certain phrase and is calculated as followed:

$$Px_\theta(x_1...x_n) = -log_2 Pr(x_1...x_n|f_\theta) = \sum_{i=1}^{n} -log_2 Pr(x_i|f_\theta(x_1...x_{i-1}))$$

However, as my project aims to fine-tune instead of train from scratch, I opted instead to fine-tune only on a set of three different canary phrases copied eight times for the model to see a sufficient number of examples. These canary phrases were "My social security number is 078051120", "My special number is 078051120", and "Here is my social security number is 078051120". The King Lear dataset is used to establish a baseline behavior while the canary dataset is used to analyze memorization.

## Model Architecture

The GRU model's architecture contains 4 million parameters and consists of 3 primary layers: an embedding layer of Shape (8, None, 256), a GRU layer of shape (8, None, 1024), and a dense layer (8, None, 86). Functionally, this model's embedding layer takes up to 256 characters for training, learns weights through 8 by 1024 GRU units, and classifies this data into 86 characters. Each categorized class is associated with a character in the vocabulary set of the Shakespeare dataset. The vocabulary consists of all alphabetical characters (both lower and upper case), numerical characters, and a set of special characters (@$(,[_ #'/;?"&*.:]!%)-). The loss function I use is the standard Categorical Cross Entropy, opting for the Sparse Categorical Cross Entropy version from Keras. Cross Entropy Loss is calculated as the sums of the entropy of each class:

$$L(x, y, \theta) = H(p, q) = -\sum_{\forall x} p(x) log(q(x))$$

Stochastic Gradient Descent (SGD) and Differential Private Stochastic Gradient Descent (DP-SGD) are used as optimizers. SGD is an optimization algorithm designed to update the gradients within a machine learning model to reach some set of weights that exists in the minima of the solution plane of an optimization problem. This updating process is calculated by setting the weights of the model $\theta_{new} \leftarrow \theta_{old} - \eta \frac{1}{m'} \sum_{j=1}^{m'} \nabla_\theta L(\bar{x}_j, \bar{y}_j, \theta)$.

DP-SGD is an optimization algorithm that incorporates differential privacy principles into the standard stochastic gradient descent procedure to protect the privacy of individual data points while allowing for useful analysis of the dataset. In each iteration of DP-SGD, gradients of the loss function with respect to the model parameters are computed for a randomly sampled mini-batch of data points. To prevent individual data points from having a disproportionate effect on the gradient updates, each gradient is clipped to a maximum L2 norm, bounding the sensitivity of the gradients. Random noise, typically drawn from a Gaussian or Laplace distribution with zero mean and a scale parameter that depends on the desired privacy level, is then calculated and added to the clipped gradients. Finally, the model parameters are updated using the noisy gradients according to the SGD update rule. This process ensures that the gradient updates incorporate a degree of uncertainty, protecting the privacy of individual data points, but comes at the cost of reduced utility and model accuracy, creating a trade-off between privacy and utility that must be carefully balanced when using DP-SGD.

### Approach

To ensure the number of examples fed to the model is the same as the canary dataset, the first 24 training data points and 24 validation data points are selected from their respective datasets and used. For my experiments, I generated 40 characters following the phrase above, such that I can follow the changes to characters at the beginning of and further within sequences. If memorization happens perfectly, the phrase memorized should be “My social security number is 078” followed by the characters ”051120My special number is 078051120My s”. These characters should appear as the encoding window of the model is inherently 256 and will take that length sequence from the 320 character canary dataset to train.

To analyze memorization in the context of fine-tuning quantatively, I track four different metrics - train loss, test loss, canary loss and log perplexity. The train and test loss are calculated using the loss function and give a metric to how accurate the model functions when prompted for that dataset. The canary loss is similar but is measured in accuracy on the canary dataset. Log perplexity is calculated by first using "My social security number is 078" to prime the model weights and using the probabilities of the successive characters "051120" in the formula above.

These metrics are used over three main sections: fine-tuning non-DP with both datasets to establish baseline performance, fine-tuning DP with various clipping norms and no noise, and fine-tuning DP with various noise amounts and no limit to clipping.

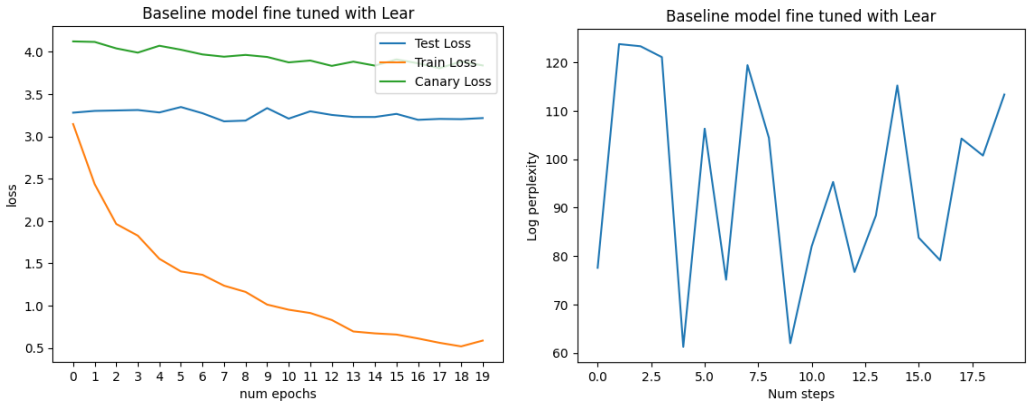
## 2 Finetuning

### 2.1 Baseline

As a baseline, I first explored how fine-tuning with the Shakespeare sub dataset, consisting of only phrases stated by King Lear, affected this loss. Generally while fine-tuning, the learning rate is set to a large value such as 0.25 and trained over a small number of epochs. However, to observe performance, I opted to set the learning rate to 0.05 and trained the model over over 20 epochs to observe performance over smaller steps. During fine-tuning, I set four different callbacks to track the train loss, test loss, canary loss and log perplexity metrics. These baseline results are plotted below in figure 1a and 1b..

As expected, the training loss decreased to a minimum and maintains that minimum at 0.5, but the test loss exhibits no change. The lack of change could be the result of the model not overfitting on the training data as it is not able to predict the Lear test set very well. Surprisingly, the canary loss depicts a slight decrease, but is too small to be conclusive. No gatherable findings for the log perplexity were concluded as the log perplexity seemed to be inconsistent. I also track how the generated text changes over each step as well. The model generates text by inputting a phrase (in this case “My social security number is 078”), sampling values from the predicted probabilities of each character type, and converting the indices of those predicted classes to their associated characters. One such example is “My social security number is 0784. Lorry’s, swear mind when Mr. Lorry’s” after fine-tuning the model for 10 epochs. The result is vaguely Shakespearian and similar to King Lear’s text.

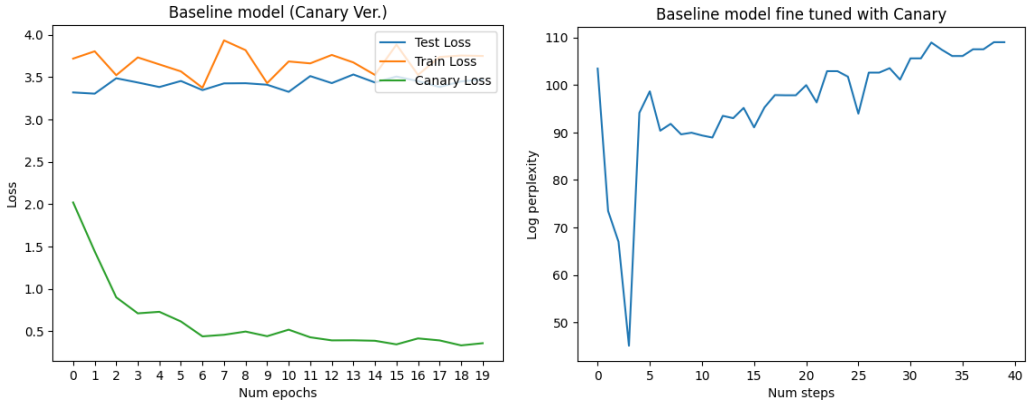
After having an established baseline performance for fine-tuning on Lear data, a baseline for fine-tuning with the




(a) Loss over epochs

(b) Log perplexity over steps

Figure 1: Loss and log perplexity over time fine-tuned with Lear




(a) Loss over epochs

(b) Log perplexity over steps

Figure 2: Loss and log perplexity over time fine-tuned with Lear

canary dataset was established. The optimizer and loss are set the same as the Lear baseline. The baseline model trained on the canary dataset has its results plotted above in figures 2a and 2b.

The canary loss decreased quicker than the baseline Lear training loss possibly because similar phrases are repeatedly shown to the model. The log perplexity spiked at two steps, but dropped to 70 before slowly increasing for the rest of the steps. Surprisingly, the log perplexity does not decrease further as the model fits to the training data. Changes to the generated text can be observed as well. Unrolling 40 characters, the model generated the phrase "My social security number is 078.821 Nocoming human creatured security a" at epoch 1. As the model approach epoch 5 during finetuning, similar values to the canary dataset result, "My social security number is 07805112M0T seeing it made", and "My social security number is 078051120My social security number is 07805" at epoch 8. As the canary social security number is 078051120 and the fine-tuned model outputs exactly that number, memorization occurred quickly as the model is able to generate almost accurate sequences after 5 steps, and exactly accurate sequences after eight.

## 2.2 DP fine-tuning on Lear dataset

Now that baseline performance is established for both finetuning on the Lear and the canary dataset, I applied DP-SGD to the model and fine-tuned it on the Lear dataset. DP-SGD primarily depends on four values–the maximum $L_2$ norm for clipping, noise multiplier, number of microbatches, and learning rate. The first two values, the clipping threshold and noise multiplier, affect both the epsilon and delta parameters of how private the DP-SGD training is. As such, I set the clipping values [0, 0.1, 0.4, 0.7, 1, 1.2, 2, 4] and the noise multiplier to 0 to observe how the model changes with different clipping thresholds. The number of microbatches is set to 1 and the learning rate is set to 0.05 to be able to observe changes gradually. I analyzed the same four metrics over each of these clipping values and plotted each set of metrics separately. After I set the clipping value to 100, such that there is effectively no clipping to the gradients during DP-SGD, I analyzed the same four metrics over the noise multipliers [0, 0.0005, .001, 0.0015, 0.002, 0.0025, .003]. The resulting plots can be found below in figure 3.

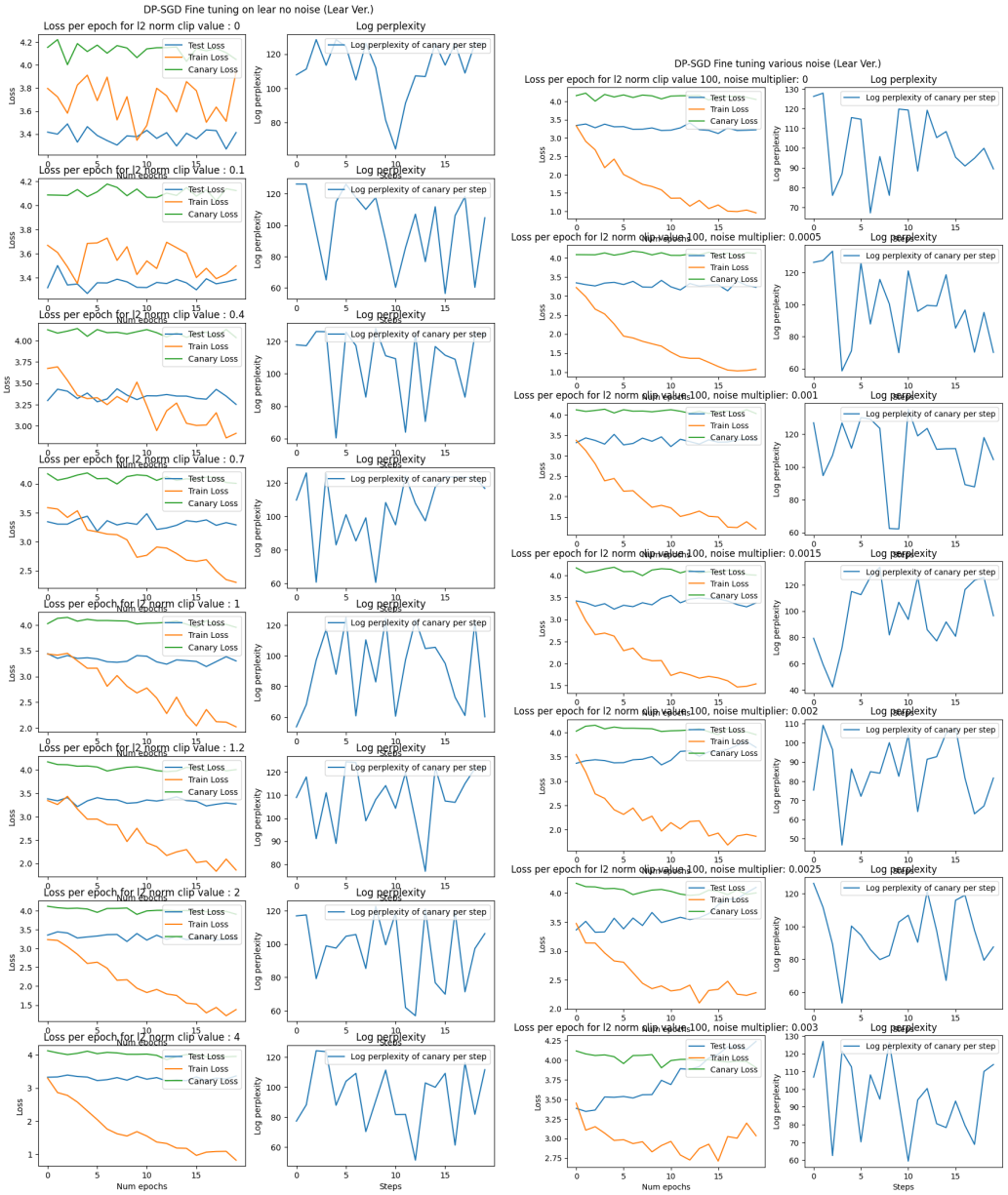
### 2.2.1 Clipping

With small clipping values, the model does not learn quickly and therefore the train loss does not change at all. This behavior is expected as the weight gradients from the learned data are over-clipped and the resulting weights update does not change the model enough to achieve a similar loss to the baseline model. With larger clipping values, the model is able to achieve the baseline loss values and generates Lear-esque text.

### 2.2.2 Noise

The train loss seemed to reach a limit of 1.0 over all noise multipliers, higher than the baseline's minimum loss. At higher noise multipliers, the achieved minimum loss is 1.5. Any step beyond the 18th increased the loss, possibly because of the noise interfering with the training. The log perplexity of the canary appeared to be similar to the baseline, such that the log perplexity values are random.

The phrases generated are similar to those from the Lear training text with lower noise multipliers, "My social security number is 078. But I was confident.\n\nThe chateau al", but become more distorted as the accumulated noise becomes larger, "My social security number is 078rs.\nThOur: siecess, impathed he, with \n". This distortion is the result of the noise interfering with pretrained weights, adjusting those weights randomly based on the sampled noise.




(a) Experiments involving various clipping values (b) Experiments involving various noise multipliers

Figure 3: DP fine-tuning with the Lear dataset

## 2.3 DP fine-tuning on Canary dataset

The same model was fine-tuned with the canary dataset setting the clipping values to [0.1, 0.3, 0.5, 0.7, 0.9] with the noise multiplier set to 0 during experiments to understand the effects of clipping. The clipping values were set to a smaller scale as with larger clipping values would result in plots that exhibit data that were too similar. Clipping values to 100 to artificially impose a no-limit to the clipping and tried noise multipliers between [0, 0.0005, .001, 0.0015, 0.002, 0.0025, .003] to observe the effects of noise. The results to these experiments can be found below in figure 4a and 4b.

### 2.3.1 Clipping

It appears that the smaller the clipping norm, the worse the model performed in terms of canary loss. Log perplexity did not change much; the log perplexity was irregular but appeared to have a slight downwards trend with a larger $L_2$ norm. This trend possibly could be explained by the minibatching and clipping of the data during DP-SGD. The canary phrase can be different from the one above as minibatching's random sampling could change the canary phrase inputted into the fine-tuning process. In other words, minibatching would randomly sample from 24 data points consisting of three canary phrases, and the resulting data can be a different 40 character sequence than above (i.e. "My social security number is 078051120My social security number is 07805" could be "My special number is 078051120My special number is 078051120Here" selecting for three different canary phrases instead of two of the same ). As such, this random sampling could result in examples different from the intended canary phrase used to prime the log perplexity calculation function; the result is a very different log perplexity of that step.

### 2.3.2 Noise

After, I set the clipping values to 100 to artificially impose a no-limit to the clipping and tried noise multipliers between [0.1, 0.3, 0.5, 0.7, 0.9] to observe the effects of noise. As a general trend, the canary loss decreased with more steps. With all noise multipliers, the test loss and the train loss increased slowly, possibly because of the noise interfering with the weights not relevant to the memorization of the canary. With 0 noise multiplier, the canary loss does not reach 0.5 as the baseline does possibly because of the effect of random sampling micro batches from the training dataset during DP-SGD. The higher the noise multiplier, the quicker the canary loss decreased to 0.5 as well. With a noise multiplier of 0.0025, the canary loss began to increase with later epochs (18-20), similarly to how the Lear train loss does with the DP fine-tuned Lear model with noise. The log perplexity appeared to follow the trend of the canary loss within the canary baseline with lower noise multipliers, and loosely with higher noise multipliers. This behavior is also expected given that with more noise there is more interference with the model's learning of the phrase; the log perplexity of the canary should be more inconsistent.

Generated text also becomes less readable as the noise multiplier increases. With 0 noise multiplier, the model is able to almost consistently generate the canary at the 5th epoch, "My social security number is 078051121My springing Pross's blows.", and fully is able to generate the canary at the 9th, "My social security number is 078051120My social security number is 07805". With the 0.0005 noise multiplier, the model still is consistently able to generate the canary. At 0.001 noise multiplier, the model returned the correct following numbers, but the noise interfered with the memorization of the rest of the phrase, resulting in phrases that look like "My social security number is 07805112020My spility number isclort. Ascin" or "My social security number is 078041110Mry socked in it."\n "My elder br". This behavior manifested in higher noise multipliers, but gradually distorted the pretrained behavior as well. For instance, with a noise multiplier of 0.003, the phrase "My social security number is 078xod dut numby sonoagger nume it sycurily" is generated at epoch 16. These words are vaguely similar to the words within the canary phrase (i.e. "numby" possibly could be "number", or "sycurily" possibly could be "security") but with any larger noise multiplier, the noise becomes too large for the model to generate meaningful text.


(a) Experiments involving various clipping values (b) Experiments involving various noise multipliers

Figure 4: DP fine-tuning with the canary dataset

# 3 Conclusion

Overall, DP-SGD as a training method for fine-tuning does seem to be a possible countermeasure against memorization with some limitations. Clipping by itself appears to be able to offer some kind of resistance to memorization in this case. Occasionally, the clipping model is able to generate artificial social security numbers that are almost entirely different – "0786215141" is generated with clipping norm 0.1. This example alludes to the possibility of clipping alone being able to generate fake sensitive information. In larger multipliers, adding noise during DP-SGD fine-tuning interferes with memorization as a whole. Even so, noise as a countermeasure also seems promising, as a model trained with a smaller noise

multiplier is able to generate entirely artificial social security numbers such as ”0780318020”. One of the main limitations to analysis was the fact that the three unique canary dataset phrases were too different to perfectly map the effects of random sampling and clipping on fine-tuning. Using one canary phrase to make DP-SGD’s minibatch random sampling more deterministic would allow for easier exploration of its effect on both weight updates and the logits responsible for text generation. To fully understand the effects of noise and clipping however, further exploration and experimentation must be done to elucidate their effects on the logits of text generation is needed. As such, there are three major points of interest for future work in the context of privacy–exploration of minibatch random sampling, effects of noise addition on text generation logits, and the combination of both.

# References


Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Caldas, S., Duddu, S. M. K., Wu, P., Li, T., Konečný, J., McMahan, H. B., Smith, V., and Talwalkar, A. (2019). Leaf: A benchmark for federated settings.

Carlini, N., Liu, C., Úlfar Erlingsson, Kos, J., and Song, D. (2019). The secret sharer: Evaluating and testing unintended memorization in neural networks.

McMahan, H. B., Ramage, D., Talwar, K., and Zhang, L. (2018). Learning differentially private recurrent language models.