Using the Secret Sharer Attack to Learn Locker Combinations from Text Messages

Boston University - CS 591 S1

John C. Merfeld

December 16, 2018

1 Motivation

The progress of real-time text prediction has exploded in recent years, with iPhone keyboards using "QuickType" since iOS 8 (mid-2014) and Google introducing "Smart Compose" for Gmail in May 2018. These text models (only two prominent examples in a rapidly growing domain) seem to learn the particular typing habits of their users. In order to be generally robust, however, they were clearly trained on an enormous body of emails and messages sent by users.

From a data privacy standpoint, there are obvious risks for anyone who has ever sent a message containing sensitive information that was later used to train a predictive text model. Is it possible to extract such secrets with mere black-box access to the model? This is the subject of a 2018 paper by Carlini et al. (hereafter referred to as "the Carlini paper"), which gives a framework for measuring the degree to which text models **memorize** their inputs, and how easily these inputs can be extracted by an attacker. It should be noted that such memorization is not the intended behavior of a predictive model. It is generalization, not memorization, that makes a model useful. That extraction is a bug and not a feature should be self-evident.

This report documents an attempt to replicate and expand on the findings in the Carlini paper. We will give a brief technical introduction, discuss how the experiments were conducted, and share the results. For a discussion of the technical specifications of the project, as well as how they from those in the Carlini paper, see the README accompanying the code at https://github.com/johncmerfeld/secretSharer.

2 Background & Definitions

2.1 Models

The models referred to in this report are **neural networks**. Thankfully, the reader requires no fundamental understanding of how neural networks are constructed. Suffice to say that we will denote specific models with f_θ, where f is some arbitrary function (computed on text input, in this case) and θ is a set of weights that resulted from particular training on a particular data set. In this report, every trained model will have the same use case and structure, as demanded by f, but the weights that affect their individual outputs are determined by different values of θ. Thus, when any variable below includes a θ subscript, that variable depends on the data and training regimen that built that particular model.

Some introductory notation will help illuminate the goals of the study. Given a model f_θ, we can measure the likelihood that a given sequence x is predicted by that model. This report will refer to that quantity as **log-perplexity**, denoted as Px_θ and defined as follows:

$$\text{Px}_\theta(x_1, \ldots, x_n) \equiv \sum_{i=1}^{n} \Big(-\log_2 \mathbf{Pr}(x_i | f_\theta(x_1, \ldots, x_{i-1})) \Big)$$

Essentially, we have defined the log-perplexity of the sequence x as the negative of the

summed probabilities of each constituent part of x, conditional on the preceding parts of x having been predicted. Note that $\text{Px}_\theta(x)$ varies with both the sequence itself and the weights from the model making the predictions. One model, $f_{\theta 1}$, might have a very low log-perplexity for a certain x, meaning it is predicted with high probability. But another model, $f_{\theta 2}$, might practically never output x.

2.2 Secrets

A secret is an intuitive concept, but it can also have a straightforward mathematical representation. In this report, we will use the notation $s[r]$ to describe a secret, where s is called the **template** and r is called the **randomness**. Consider the secret "My birthday is 03-15." Here, the format s is "My birthday is _-_," and the randomness r is 03-15. We refer to r as being drawn from a **randomness space** denoted by $\mathcal{R}$. In this example, $\mathcal{R}$ is the set of all birthdays, and it has size 366 (because of leap years).

Consider another example, where $s =$ "Here's my social security number: _ _ _-_ _-_ _ _ _" and $\mathcal{R}$ is the set of 10-digit numbers. Clearly, any particular $s[r]$ would be difficult to guess correctly at random.

2.3 Exposure

Suppose we have a model f_θ and give it s as input. What will it predict? Out of the randomness space $\mathcal{R}$, which instance r does f_θ output? We address such questions with a quantity called **rank**. As with log-perplexity, rank is defined in terms of a model f_θ, and a sequence x, except in this case, x is an instance of a secret, $s[r]$. Rank is defined this way:

$$\textbf{rank}_\theta(s[r]) \equiv |\{r' \in \mathcal{R} : \text{Px}_\theta(s[r']) \leq \text{Px}_\theta(s[r])\}|$$

So the rank of a secret for a given model measures how many secrets in the randomness space are predicted at least as often as the one in question. A rank of 1 indicates that a particular value of r has the lowest log-perplexity of any instance of a secret with format s. In such a case, if $s[r]$ were included in the data set that trained f_θ, we would conclude that the model has **completely memorized** the secret, and it would be easy to extract. To calculate the rank of a secret, of course, we must enumerate the log-perplexities of every possible secret on the model, which can be computationally prohibitive.

The Carlini paper defines a final quantity called **exposure** to measure the degree to which a secret is at risk of being extracted, given black box access to a model.

$$\textbf{exposure}_\theta(s[r]) \equiv -\log_2 \textbf{Pr}_{t \in \mathcal{R}}\Big[\text{Px}_\theta(s[t]) \leq \text{Px}_\theta(s[r])\Big] = \log_2|\mathcal{R}| - \log_2 \textbf{rank}_\theta(s[r])$$

Exposure is an attractive quantity because it is positively defined (greater values of exposure have higher numeric values) and it can be estimated efficiently without enumerating over every possible secret (although this benefit is beyond the scope of this report). Note that exposure is upper-bounded by $\log_2|\mathcal{R}|$, the log of the size of the randomness space. Thus,

we will sometimes instead refer to relative exposure, when comparing secrets with different randomness spaces.

This report will use exposure as its primary metric for evaluating models' memorization of secrets. Our procedure is to build different models $\{f_\theta\}$ under a variety of conditions to understand which factors impact the memorization, exposure, and extraction of secrets.

3 Experimental Setup

3.1 The data

The model is trained on a corpus of nearly 60,000 SMS (text) messages collected by researchers at the National University of Singapore. A publicly available corpus, it is available for download from Github as an XML file.

The text messages in their raw form are difficult to process as natural language due to their widespread use of nonstandard spelling and punctuation. For example, the model should know that "coz," "cuz," and "cos" are sometimes used in place of the word "cause," and that "ll c u wid dm at lect lar" should be read as "i will see you with them at lecture later." Over 200 different regular expression instructions are used to standardize the messages. These commands include removing laughter, separating distinct words with only punctuation (and no whitespace) between them, and generally working to ensure that fully spelled out words are grouped together with their abbreviated counterparts.

3.2 The model

Once the data are cleaned, they are split into training, validation, and testing subsets and used to build a **word-level generative sequence model with fixed-size windowing**. The reader is owed some explanation of these terms:

- **Word-level** means that words are the atomic units that the model trains and predicts on. Such a model, given the input "Don't put all your eggs in one _____" might predict "basket." This is opposed to a **character-level model**, which, given the input "MISSISSIPP_" might predict "I."
- **Generative sequence** broadly means that the model predicts the next word in an ordered set (as above), as opposed to classifying a word or sentence into some category.
- **Fixed-size windowing** means that the model trains and predicts on sets of k words at a time. These may overlap during training. For example, if $k = 5$, the sentence above would be split into "Don't put all your eggs," "put all your eggs in," "all your eggs in one," etc. This is opposed to **stateful input processing**, which makes predictions based on the overall "state" of the entire text input.

In summary, the model under discussion is given a sequence of $k - 1$ words and tries to predict which word should come next.

3.3 Calculating exposure

In order to measure the model's degree of memorization, we insert one or more copies of a secret $s[r]$ into the training data. The format s can be anything, as long as its length is $k-1$ words or greater (because the model must make a prediction from the entire format as input). After the model is trained, we measure the probability of every possible secret within the randomness space to which r belongs, and find the rank of $s[r]$. From there, we calculate and record the exposure. If it is maximal within the randomness space, we consider the secret ot be "extracted."

3.4 Hyperparameters

The Carlini paper confirmed some intuitive hunches about which experimental setups would make extraction more likely. These included allowing the model to look at the training data more times and inserting the secret with greater frequency into the data. In order to test the limits of model memorization, our experiments studied the following hyperparameters. **Standard values** refer to the usual value of the parameter held constant while others are varied.

- The **Insertion rates of "true" and "false" secrets** refer to how many times $s[\hat{r}]$ and $s[\text{rand}(t)]$, respectively, are inserted into the training data. Here, $s[\hat{r}]$ is the secret whose exposure we are measuring each $s[\text{rand}(t)]$ is an individually random secret of the same format. **Standard values:** 1 true secret and 0 false secrets.

- The **training epochs and batch size** describe how long the model trained on the data. The epoch number is essentially how many times the model sees each record, and the batch size is essentially how many records the model views at once (and can be thought of as the step size of a gradient descent). More epochs and a lower batch size increase the time it takes to train the model. **Standard values:** Batch size of 512.

- The **distinct values and prefix length** are qualities of the secret. If the prefix length (or $|s|$) is 4, the data windows discussed above will be of size 5, where the fifth word is the beginning of r. The number of distinct values refers to how many different values each slot of r can take. For a credit card number, for example, this would be 10. **Standard values:** 500 distinct values and prefix length 4.

4 Results

In all, 128 of models are trained and attacked under this variety of conditions. At best, they achieve a test accuracy of about 12.5% (1 in 8 words predicted correctly). Their exposures are calculated using the method above and normalized so that the maximum possible exposure is set to 1. The raw results are available as a CSV along with the code.

Prefix length	Prefix text	# of trials	Avg relative exposure
3	“my combination is ”	5	0.42
4	“my locker combination is ”	69	0.57
5	“my secret locker combination is ”	8	0.64
6	“my super secret locker combination is ”	5	0.42
7	“my super secret locker combination might be ”	5	0.42
8	“dont tell anyone but my locker combination is ”	5	0.58

Table 1: There is no obvious relationship between the secret prefix (which determines the window size of the model) and the exposure

Analysis on the experimental results shows that most of the hyperparameters almost certainly have no significant effect on the exposure. For example, **Table 1** shows that there is no significant relationship between the window length of the model and the exposure. The same is true of the ratio of true to false secrets, at least in the ranges in which they were tested here. In fact, the number of epochs appears to be the primary determinant of exposure, as **Figure 1** demonstrates.

Although the Carlini paper found that a smaller batch size decreased exposure, we found no significant relationship, at least in the range of batch sizes tested. If anything, smaller batch sizes may have increased exposure slightly, but there were insufficient resources to test very small batches.

It is worth noting, as the Carlini paper did, that the increase in exposure begins will before the model starts to overfit on the training data. We see in **Figure 2** how the validation accuracy's progress begins to diverge from the training accuracy after around 15 epochs, but many models exposed their secrets completely after 10 or fewer epochs.

Probably the most interesting result is what took place on the extreme low ends of secret insertion and training time. The Carlini paper showed that memorization occurred well before any overfitting began to happen, and our results agree with this. Overfitting (an increase in training accuracy while validation accuracy began to decrease) usually began around epoch 25, and most models were not trained nearly that long. On the contrary, some level exposure increase occurs almost immediately, and if the secret is inserted many times, it is at risk of extraction after just a few epochs. This finding is not totally symmetric; that is, even if the secret is only inserted once into a vast data set, it can still be perfectly recovered if there are enough training epochs. **Figure 3** illustrates this.

5 Conclusions

Ultimately, these findings serve primarily as a confirmation of the results of the Carlini paper; they are probably most notable for the negative results and lack of correlation between several hyperparamters and the exposure of secrets. Notably, we were able to reproduce exposure after only a few training epochs even though our model used fixed-size windowing at a word level and most of the Carlini results are from a state-based character model.

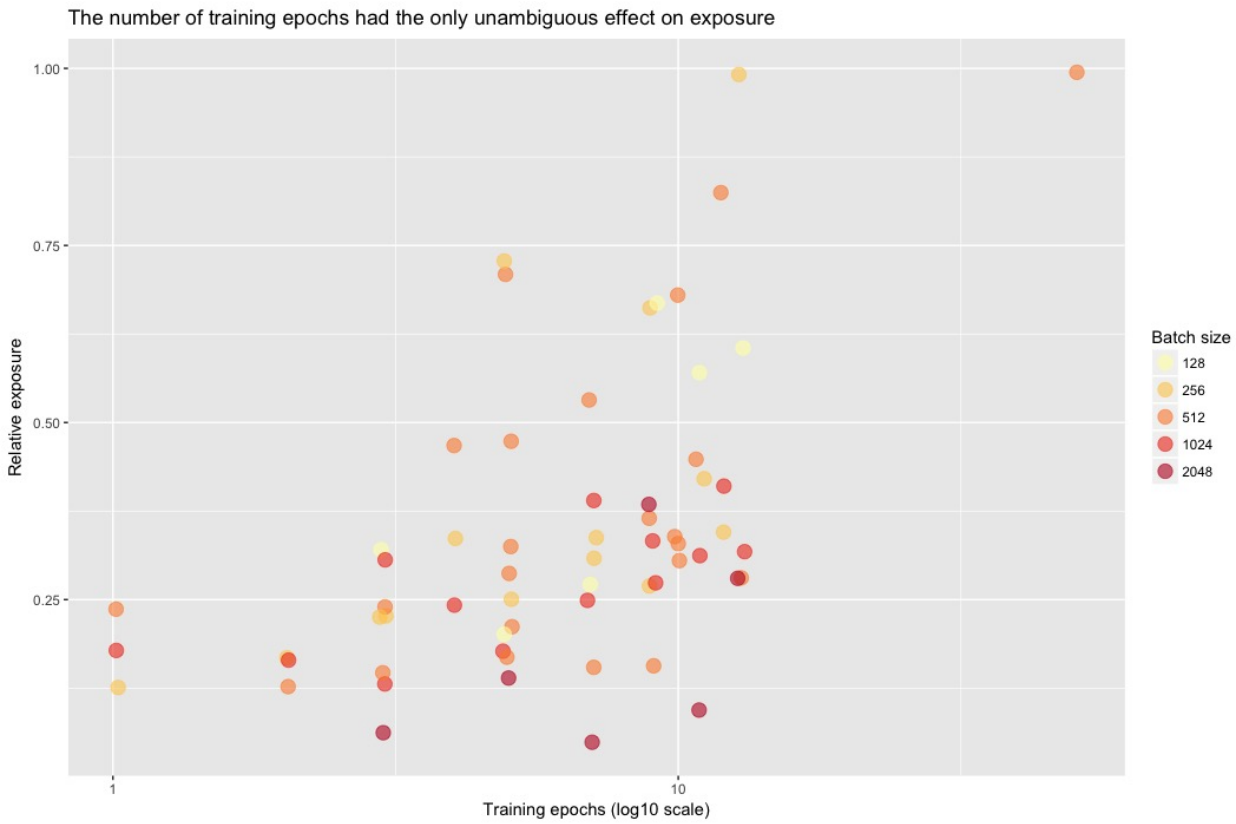

Figure 1: There is a clear correlation between training epochs and exposure. Batch size does not seem to have any such effect. Note that a relative exposure value of 1 means the secret was extracted. Standard values (see above) used for other parameters.

The topline takeaway for the author is how quickly exposure increases with training epochs, despite low insertion rate and the noisy secrets in the training data. It suggests that deep learning models might be able to memorize such information even if they were not fed it explicitly, but are instead overfitted on some combination of records that an attacker could tease out. Given better hardware, more time, and proper coding skills, the author would have liked to explore these more implicit insertion cases more thoroughly.

Once the math is understood, exposure is a fairly intuitive way to measure privacy loss in a deep learning model. One major limitation of our study and the Carlini paper is their measurement against "toy" models, by which we mean models built expressly for the purposes of the experiment. In order to gauge the effects and consequences of memorization "in the wild," it would need to be tested on black-box models provided by Google, Amazon, etc.

But perhaps of greater danger are language models built by smaller firms whose data scientists may lack the expertise to properly sanitize and privatize the data. As this report reveals, it does not take an experienced machine learning engineer to build a language model with fairly high accuracy, and while that may lead to broad utility in the short term, such

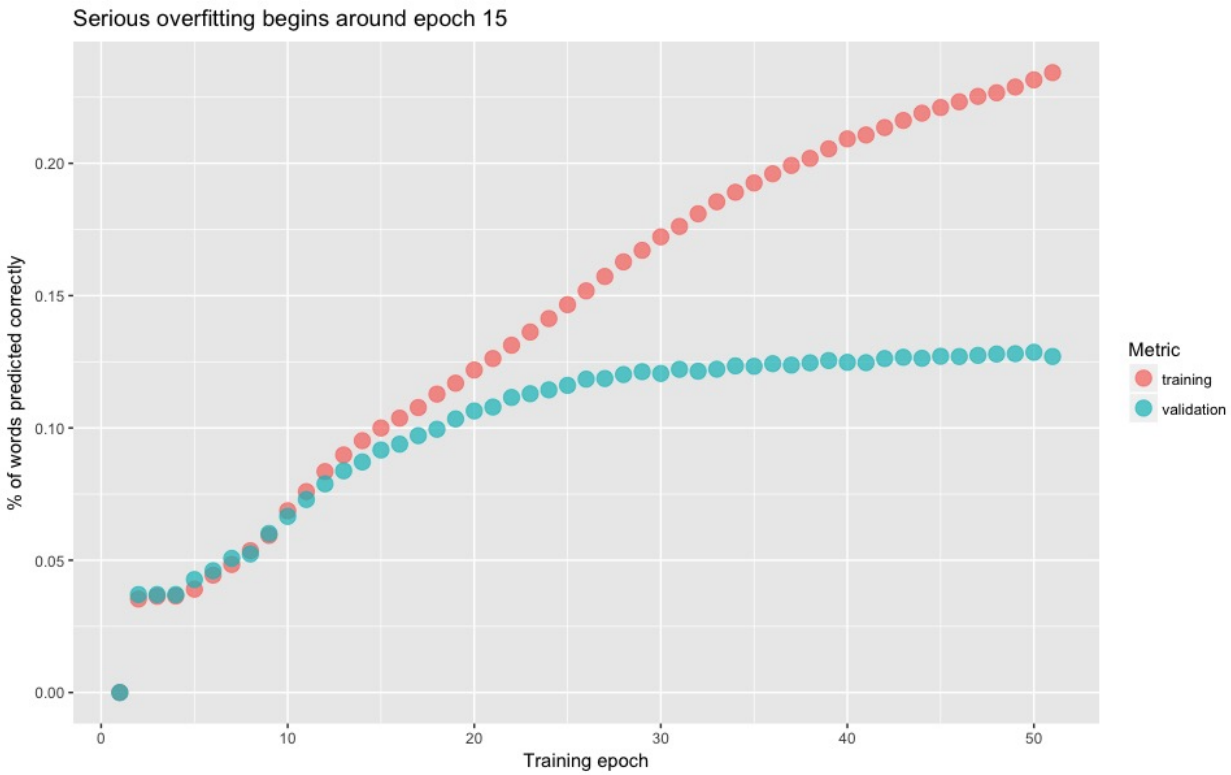

Figure 2: The training and validation accuracy begin to diverge relatively quickly, even as the validation accuracy continues to increase. Still, we see high exposure rates among models trained for fewer than 15 epochs.

technological power could allow privacy breaches if not handled with care.

5.1 Acknowledgements

Almost all of the Python code was written by the author. The TensorFlow architecture and some text cleaning code were adapted from the blog post "How to Develop a Word-Level Neural Language Model and Use it to Generate Text," which can be found at https://machinelearningmastery.com/how-to-develop-a-word-level-neural-language-model-in-keras/. A few other techniques, such as one-hot encoding and fixed-size windowing, were adapted from code posted on Stack Overflow.

The raw SMS data can be found at https://www.comp.nus.edu.sg/ kanmy/papers/smsCorpus.pdf

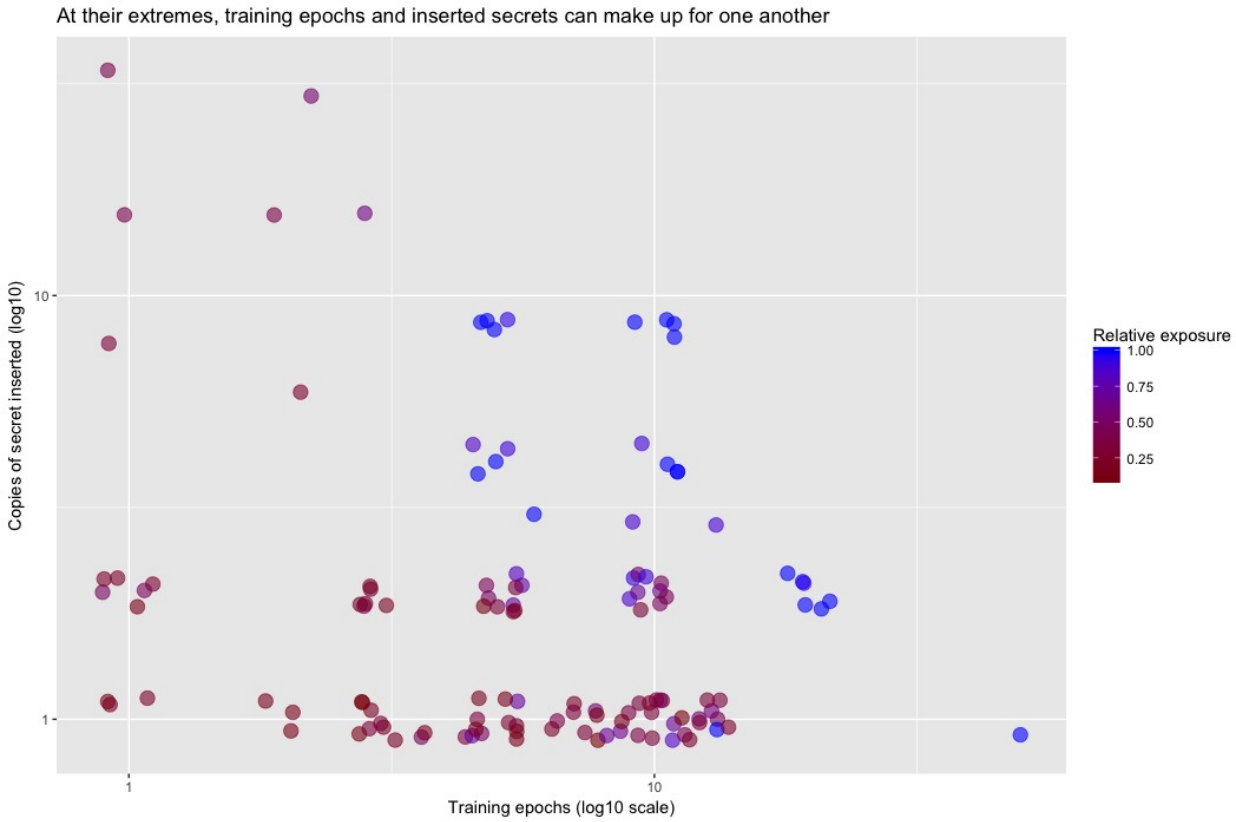

Figure 3: Models trained with very few eopchs generally need more secret insertions, but at the extreme low end, even high secret insertion does not greatly increase exposure. Low insertion can be overcome by a long training time, it appears. Standard values (see above) used for other parameters.