Differential Privacy

Recently, many organizations have released machine learning models trained on massive datasets (GPT-3, YOLO, etc...). This is a great contribution to science and streamlines modern AI research. However, publicizing these models allows for the potential ``reverse engineering'' of models to uncover the training data for the model. Specifically, an attacker can download a model, look at the parameter values and then try to reconstruct the original training data. This is particularly bad for models trained on sensetive data like health information. In this section we are going to use randomness as a method to defend against algorithmic ``reverse engineering.''

Injecting Randomness

One way to combat algorithmic reverse engineering is to add some random element to an already existing dataset. Let

$$X_1, \dots, X_i \overset{\text{i.i.d}}{\sim} \text{Bern}(p)$$

represent a set of real human data. Consider the following snippet of code:

```
def calculateXi(Xi):
    return Xi
```

Quite simply, an attacker can call the above for all 100 samples and uncover all 100 data points. Instead, we can inject an element of randomness:

```
def calculateYi(Xi):
    obfuscate = random() # Bern with parameter p=0.5
    if obfuscate:
        return indicator(random())
    else:
        return Xi
```

The attacker can in expectation call the new function 100 times and get the correct values for 50 of them (but they wont know which 50).

Recovering p

Now consider if we publish the function calculateYi, how could a researcher who is interested in the mean of the samples get useful data? They can look at:

$$Z = \sum_{n=1}^{100} Y_i.$$

Which has expectation:

$$E[Z] = E\left[\sum_{n=1}^{100} Y_i\right] = \sum_{n=1}^{100} E[Y_i] = \sum_{n=1}^{100} \left(\frac{p}{2} + \frac{1}{4}\right) = 50p + 25$$

Then to uncover an estimate, the scientist can do,

$$p \approx \frac{Z - 25}{50}$$

And proceed to conduct more research!