Differentially Private Machine Learning

Shengwen Yang and Robert Harlow

- Introduction to privacy problem in machine learning
- Two approaches to differentially private ML

Introduction to privacy problem in machine learning

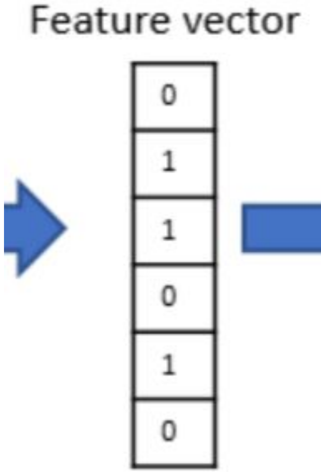
Classification task: a function *f* from examples to classes.

Ex. From images of digits to the corresponding integers.

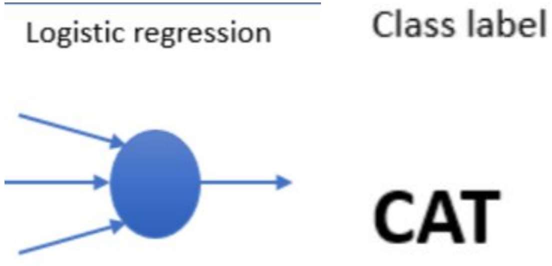
Steps:

- Data collection
- Camera - Database
- Preprocessing
- Rejecting poor data

Introduction to machine learning



- Feature extraction
  - Find relevant attributes - collect measurements in a vector
- Collecting training data
- Inference: Apply a model *g* to inputs



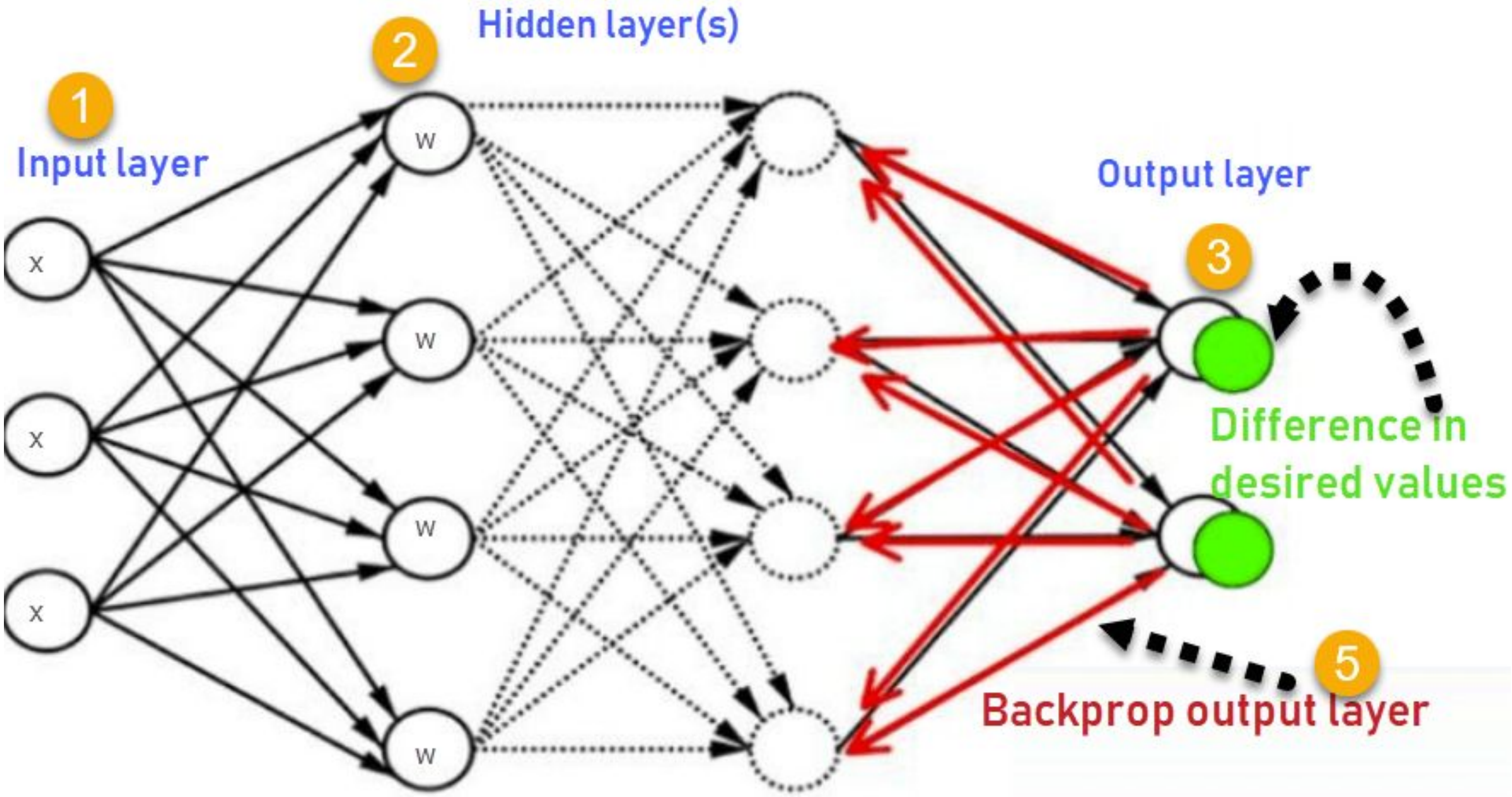
Two general categories:

1. Supervised learning

-Based on a collection of known input - output pairs (training data)

2. Unsupervised learning
   - No training data/examples (ex. clustering)

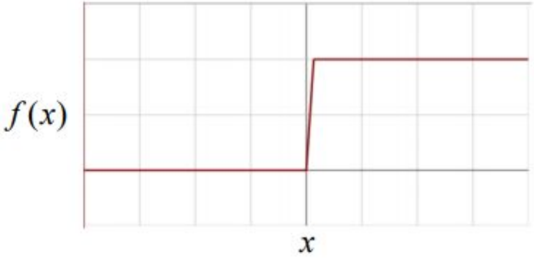
Neural Network



Activation function

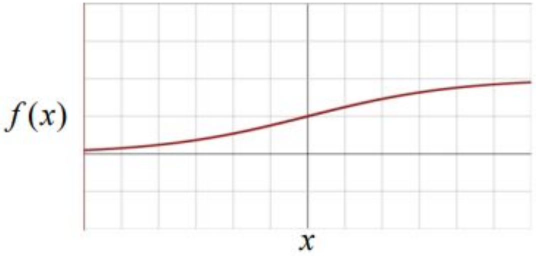
- What makes the neuron "fire"?
  - Step function

$$f(x)=\begin{cases}0 & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases}$$



  - Sigmoid function

$$f(x)=\frac{1}{1+e^{-x}}$$



  - Rectified linear unit (ReLU)

$$f(x)=\max(0,x)$$

Images from Wikipedia: Activation function

Back-Propagation Algorithm

1. Make prediction (Forward Pass)
2. Calculate the total Error (Loss), E
3. Calculate gradient of the loss function w.r.t. each weight, $\partial E/\partial w_i$
4. Update all weights by taking a step in the opposite direction: $\Delta w_i = -\alpha \, \partial E / \partial w_i$
5. Iterate

Feature Detectors

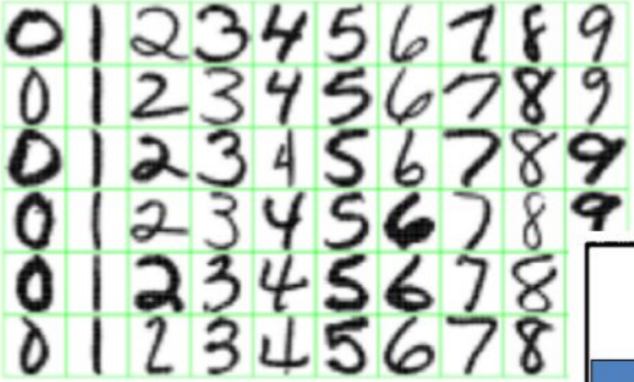
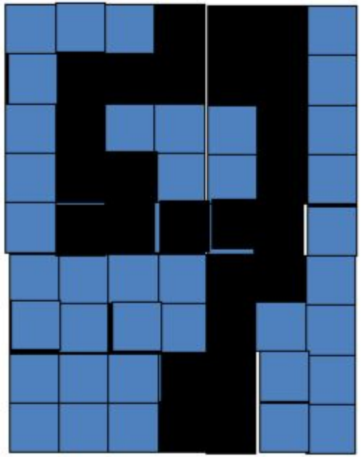


Figure 1.2: *Examples of handwritten digits from postal envelopes.*



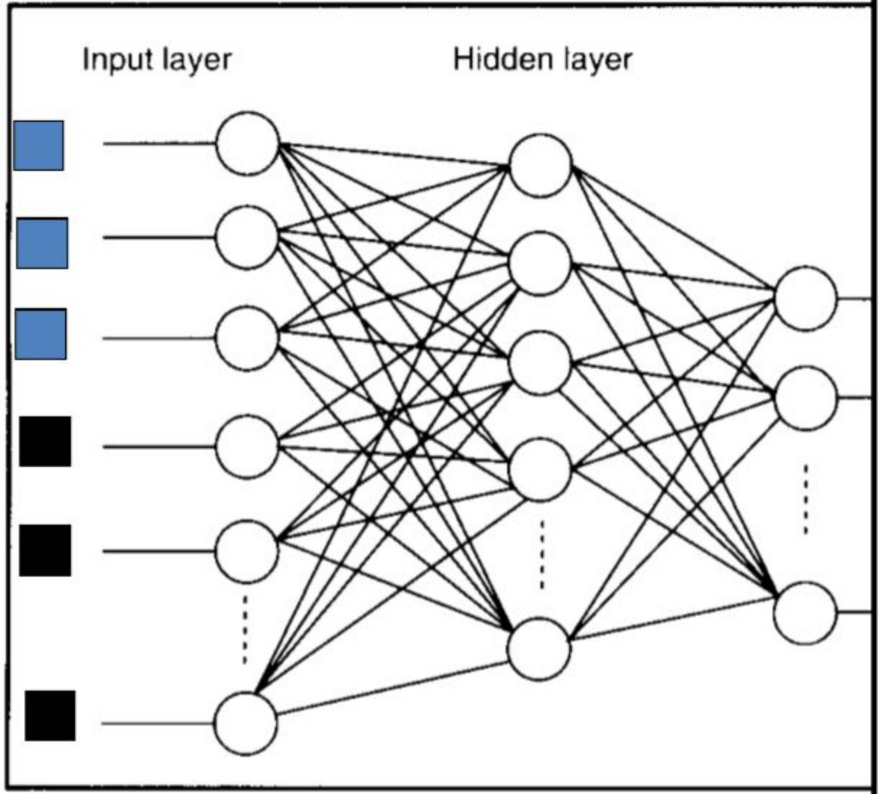
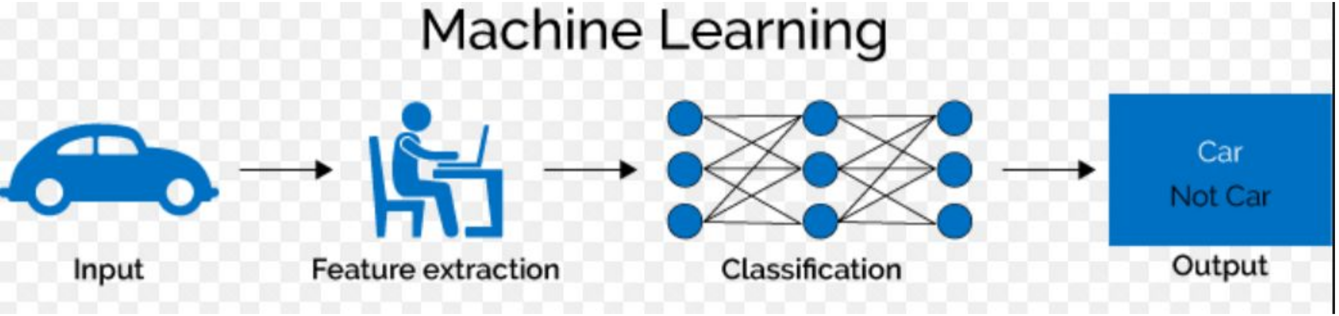


Figure 1: An image recovered using a new model inversion attack (left) and a training set image of the victim (right). The attacker is given only the person's name and access to a facial recognition system that returns a class confidence score.

Attackers’ goals and threats

Two attackers’ goals:

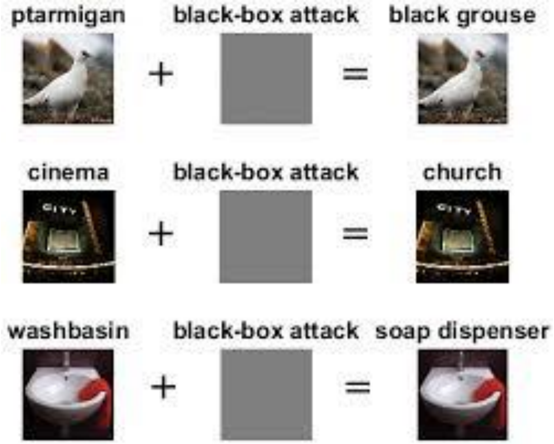
- The extraction of training data from a model
- Testing whether an input-output pair, or simply and input or an output, is part of the training data

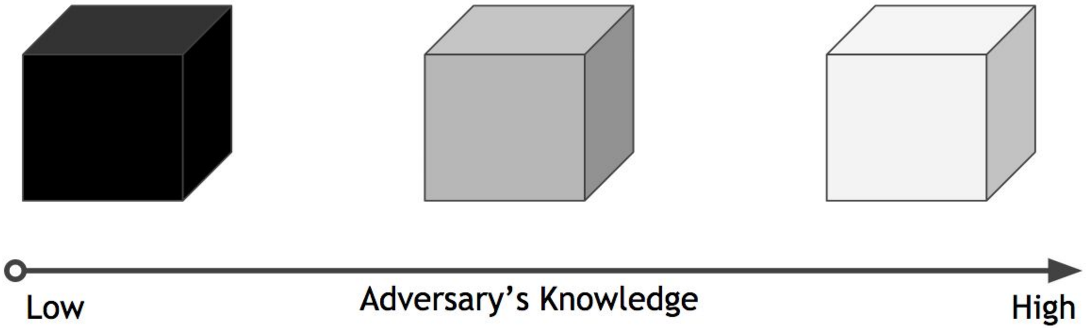


Attackers' goals and threats

Two Threats:

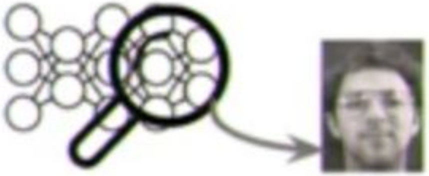
- "Black-box": attackers can apply the model *g* to new inputs of their choice, possibly up to some number of times or under other restrictions.
- "White-box": attackers can inspect the internals of the model *g*





Model querying (**black-box adversary**)

Shokri et al. (2016) *Membership Inference Attacks against ML Models*
Fredrikson et al. (2015) *Model Inversion Attacks*

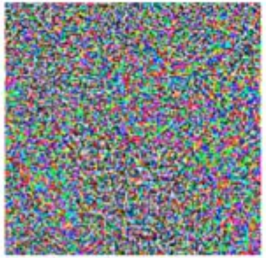


Model inspection (**white-box adversary**)

Zhang et al. (2017) *Understanding DL requires rethinking generalization*

stop sign

+ 0.001×



=

teddy bear

Our objective

We are training a model “A” on dataset “d”. A sequence of queries is made on dataset “d” before and/or during the training of model “A”.

We assume that by inspecting model “A”, an adversary could potentially recover the responses to those queries in their entirety. (the information contained in those responses may have been encoded in model “A”.)

Then, to protect the privacy of entities in dataset “d”, we must ensure that differentially private algorithms are used to transfer information from dataset “d” to model “A”.

Two approaches to differentially private ML

--Noisy Stochastic Gradient Descent (SGD)

Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, andLi Zhang. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016. [Online]. Available: http://arxiv.org/abs/1607.00133

--Private Aggregation of Teacher Ensembles (PATE)

N. Papernot, M. Abadi,´U. Erlingsson, I. J. Goodfellow, and K. Tal-war, “Semi-supervised knowledge transfer for deep learning fromprivate training data,”CoRR, vol. abs/1610.05755, 2016, presentedat the 5th International Conference on Learning Representations,2017. [Online]. Available: http://arxiv.org/abs/1610.05755

Noisy Stochastic Gradient Descent (SGD)

1. While training a model "A" (characterized by parameters theta) on dataset "d", at each step:

    -- Use a randomized algorithm "M" to choose a delta theta (dependent on the gradient determined by standard back-propagation) by which to update theta.

Back-Propagation Algorithm

Initialize the weights in the network (usually random values)

Repeat until stopping criterion is met {

forall *p*,*q* in network, $\Delta W_{p,q} = 0$

foreach example ***e*** in training set **do** {

O = neural_net_output(***network***, ***e***) // forward pass

Calculate error (***T*** - ***O***) at the output units // ***T*** = teacher output

Compute $\Delta w_{j,k}$ for all weights from hidden unit j to output unit k

Compute $\Delta w_{i,j}$ for all weights from input unit i to hidden unit j

forall *p*,*q* in network $\Delta W_{p,q} = \Delta W_{p,q} + \Delta w_{p,q}$

backward pass

}

for all *p*,*q* in network $\Delta W_{p,q} = \Delta W_{p,q}$ / num_training_examples

network = update_weights(***network***, $\Delta W_{p,q}$)

}

Note: Uses average gradient for *all* training examples when updating weights

Algorithm 1 Differentially private SGD (Outline)

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.

Initialize θ_0 randomly

for $t \in [T]$ **do**

Take a random sample L_t with sampling probability L/N

Compute gradient

For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip gradient

$\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$

Add noise

$\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$

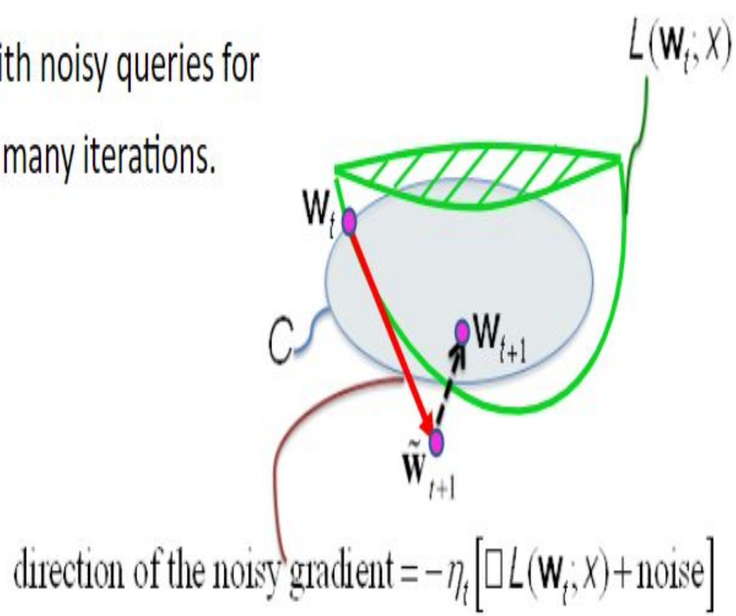
Descent

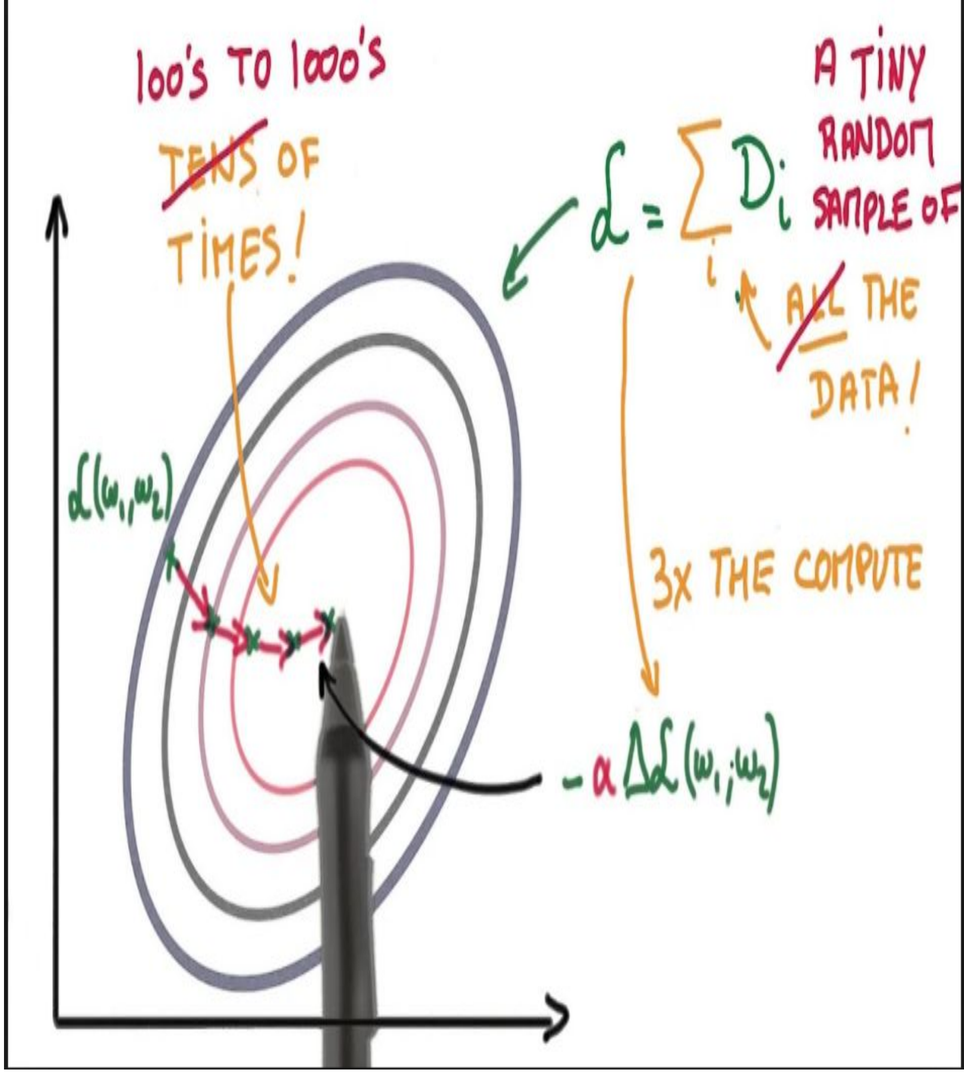
$\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$

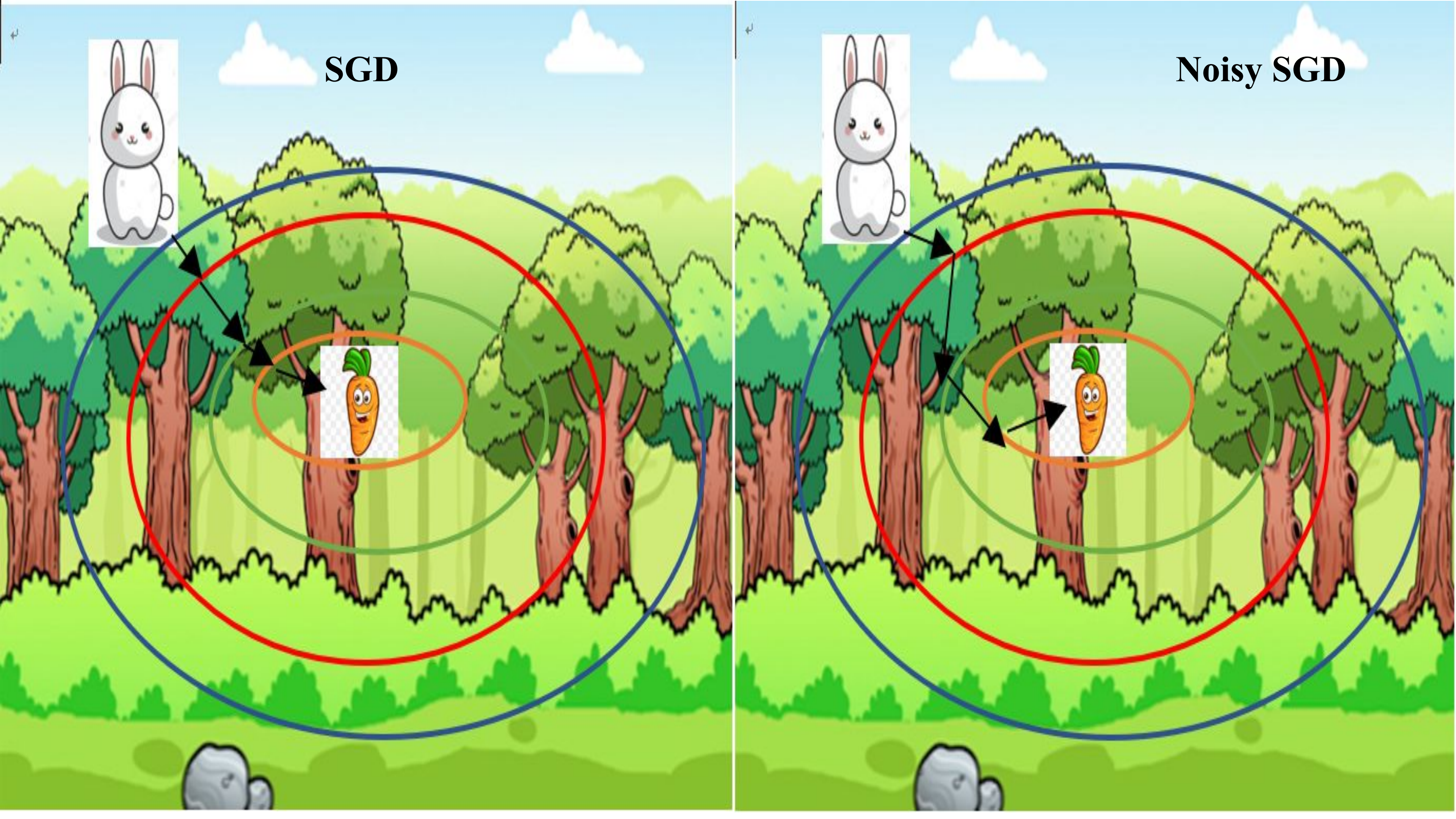
Output θ_T and compute the overall privacy cost (ε, δ) using a privacy accounting method.

Noisy stochastic gradient descent

- Run SGD with noisy queries for sufficiently many iterations.






SGD
Noisy SGD

Fisher's Iris data set

A dataset containing information on 150 distinct Iris plants from one of three species, Iris setosa, Iris versicolor, and Iris virginica. Each records contains measurements of petal length, petal width, sepal length, and sepal width, along with a species classification. Please see more here: https://en.wikipedia.org/wiki/Iris_flower_data_set

Figure 1. Iris setosa (by Radomil, CC BY-SA 3.0), Iris versicolor, (by Dlanglois, CC BY-SA 3.0), and Iris virginica (by Frank Mayfield, CC BY-SA 2.0).

Professor Calculus

To help facilitate efforts to preserve the habit of the Iris species studied in Fisher's data set, Professor Calculus wants to train a neural network model capable of accurately predicting an Iris's plants species from pedal and sepal dimensions.

Our protagonist, Iris

Iris is a Iris Setosa. She is grateful for Professor Calculus' desire to help with habit preservation, but she is also leary about participating in his research study due to privacy concerns, particularly because of...

>>>
Image: Public domain. https://en.wikipedia.org/wiki/Iris_setosa#/media/File:Iris_setosa.JPG

Our antagonist, Jack

Jack is a determined adversary with prior knowledge about our dataset. Jack's prior knowledge is:

Iris setosa = tasty.

Iris versicolor = not tasty.

Iris virginica -> stomach ache.

Fortunately Jack cannot tell different Iris species apart, so we must make every effort to protect Iris's privacy.

Figure. Scruffy Blacktail Eating (by Jessie Eastland CC BY-SA 4.0).

To protect Iris's privacy, Professor Calculus decides to use the TensorFlow Privacy library.

The Tensorflow Privacy Library.

--Uses noisy Stochastic Gradient Descent to achieve good privacy guarantees.

--Uses Renyi Differential Privacy to account for privacy lost at each training step. Privacy loss is aggregate at the end of training and recast as a (epsilon,delta) differential privacy guarantee. This process is referred to as "privacy accounting".

The library's Github repository is https://github.com/tensorflow/privacy .

Renyi Differential Privacy (RDP)

RDP defines a new notion of differential privacy (DP) based on a mathematical concept called Renyi divergence.

Mironov, I. (2017, August). Rényi differential privacy. In 2017 IEEE 30th Computer Security Foundations Symposium (CSF) (pp. 263-275). IEEE. Online at: https://arxiv.org/pdf/1702.07476.pdf

Definition of Renyi Divergence

Definition 3 (Rényi divergence)**.** For two probability distributions P and Q defined over $\mathcal{R}$, the Rényi divergence of order $\alpha > 1$ is

$$D_\alpha(P\|Q) \triangleq \frac{1}{\alpha - 1} \log \mathrm{E}_{x\sim Q} \left(\frac{P(x)}{Q(x)}\right)^\alpha.$$

(Mironov 2017)

Definition of Renyi Differential Privacy (RDP)

Definition 4 ((α, ϵ)-RDP). A randomized mechanism $f : \mathcal{D} \mapsto \mathcal{R}$ is said to have ϵ-Rényi differential privacy of order α, or (α, ϵ)-RDP for short, if for any adjacent $D, D' \in \mathcal{D}$ it holds that

$$D_\alpha (f(D) \| f(D')) \leq \epsilon.$$

(Mironov 2017)

The Privacy Accountant

The composite effect of N RDP mechanisms can be calculated by utilizing the RDP composition theorem.

Proposition 1. *Let* $f : \mathcal{D} \mapsto \mathcal{R}_1$ *be* (α, ϵ_1)*-RDP and* $g : \mathcal{R}_1 \times \mathcal{D} \mapsto \mathcal{R}_2$ *be* (α, ϵ_2)*-RDP, then the mechanism defined as* (X, Y)*, where* $X \sim f(D)$ *and* $Y \sim g(X, D)$*, satisfies* $(\alpha, \epsilon_1 + \epsilon_2)$*-RDP.*

(Mironov 2017)

Thus: **RDP Composition Theorem:** The composition of N RDP(alpha, epsilon_n) mechanisms is (alpha, Sum_n[epsilon_n]) RDP.

Privacy Accounting for the Gaussian Mechanism

Corollary 3. *If f has sensitivity 1, then the Gaussian mechanism $\mathbf{G}_\sigma f$ satisfies $(\alpha, \alpha/(2\sigma^2))$-RDP.* (for all alpha > 1)

$\mathbf{G}_\sigma f(D) = f(D) + N(0, \sigma^2)$
where $N(0, \sigma^2)$ is normally distributed random variable with standard deviation σ^2 and mean 0.

f's ℓ_2-sensitivity is defined as

$$\Delta_2 f \triangleq \max_{D,D'} \|f(D) - f(D')\|_2$$

taken over all adjacent inputs D and D'.

(Mironov 2017)

Finding an approximately optimal epsilon for desired delta.

By taking RDP(alpha, epsilon) -> DP(epsilon,delta) and by trying all values for alpha (recall that for RDP, alpha can be choosen to be any real number), it is possible to find an optimal (epsilon,delta) DP guarantee for delta fixed.

Proposition 3 (From RDP to (ϵ, δ)-DP)**.** *If f is an (α, ϵ)-RDP mechanism, it also satisfies $(\epsilon + \frac{\log 1/\delta}{\alpha - 1}, \delta)$-differential privacy for any $0 < \delta < 1$.* (Mironov 2017)

Mironov shows that for RDP(alpha,epsilon) -> DP(epsilon,delta) and delta fixed, the optimal epsilon can be approximated well by trying just a few values for alpha. (Mironov 2017)

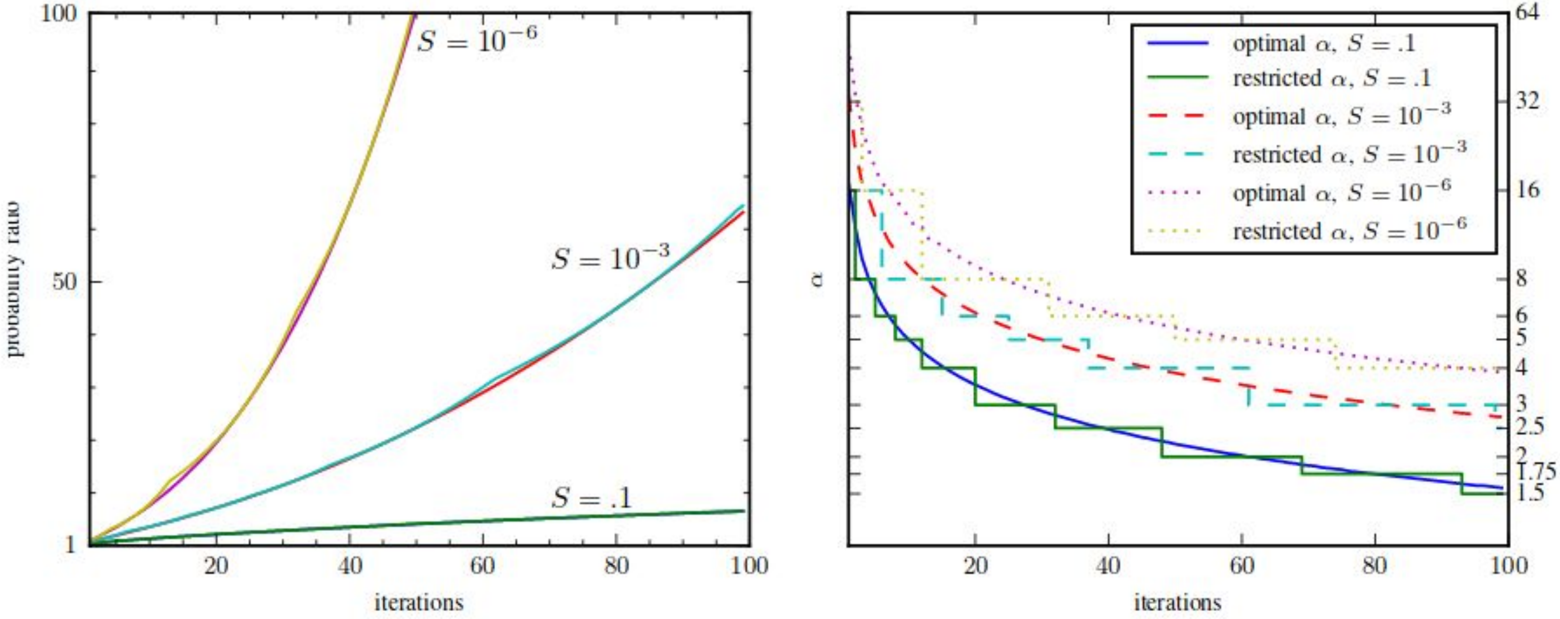


Fig. 3. Left: Bounds on the ratio $\Pr[f(D') \in S]/\Pr[f(D) \in S]$ for $\Pr[f(D) \in S] \in \{.1, 10^{-3}, 10^{-6}\}$ for up to 100 iterations of a mixed mechanism (randomized response with $p = .52$, Laplace with $\lambda = 20$ and Gaussian with $\sigma = 10$). Each bound is computed twice: once for an optimal choice of α and once for α restricted to $\{1.5, 1.75, 2, 2.5, 3, 4, 5, 6, 8, 16, 32, 64, +\infty\}$. The curves for two choices of α are nearly identical. Right: corresponding values of α in log scale.

New Hyperparameters for noisy SGD

--L2_NORM_CLIP (C)

--NOISE_MULTIPLIER Specifies how much noise is added in the add noise step.

--MICROBATCHES To increase computing speed (through parallelism), gradient clipping is done only MICROBATCHES times per batch. If MICROBATCHES is equal to BATCH_SIZE, then gradient clipping is done on a per example basis. A larger value for MICROBATCHES improves privacy performance. MICROBATCHES must evenly divide BATCH_SIZE

Algorithm 1 Differentially private SGD (Outline)

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.

Initialize θ_0 randomly

for $t \in [T]$ **do**

Take a random sample L_t with sampling probability L/N

Compute gradient

For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip gradient

$\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$

Add noise

$\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$

Descent

$\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$

Output θ_T and compute the overall privacy cost (ε, δ) using a privacy accounting method.

(Abadi et al, 2016)

Our code for the Iris Dataset

We train models on the Iris Dataset using a fully connected neural network with two hidden layers. In our code, the model architecture is defined like this:

```
############## Define model architecture. ##############
inputs = tf.placeholder(tf.float32, shape=(None, 4))
hidden_layer_1 = Dense(128, activation='relu')(inputs)
hidden_layer_2 = Dense(128, activation='relu')(hidden_layer_1)
predictions = Dense(3, activation='softmax')(hidden_layer_2)
```

How our code substitutes noisy SGD for standard SGD (Using TensorFlow Privacy)

Our Imports from TensorFlow Privacy:

```
from privacy.analysis.rdp_accountant import compute_rdp
from privacy.analysis.rdp_accountant import get_privacy_spent
from privacy.optimizers.dp_optimizer import DPGradientDescentGaussianOptimizer
```

```
############ Configure our optimizer. ############
if dp == 1:
    optimizer = DPGradientDescentGaussianOptimizer(
            l2_norm_clip=L2_NORM_CLIP,
            noise_multiplier=NOISE_MULTIPLIER,
            num_microbatches=1,
            learning_rate=learning_rate)
else:
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
```

Setting Hyperparameters

```
############### Define Hyperparameters ################
BATCH_SIZE = 1
NOISE_MULTIPLIER = 5.0
L2_NORM_CLIP = 1.0
MICROBATCHES = BATCH_SIZE #MICROBATCHES MUST evenly divide BATCH_SIZE
EPOCHS = 200
LEARNING_RATE = 0.0005
TARGET_DELTA = 1e-3
```

Implementing the “Privacy Accountant”

```
from privacy.analysis.rdp_accountant import compute_rdp
from privacy.analysis.rdp_accountant import get_privacy_spent
from privacy.optimizers.dp_optimizer import DPGradientDescentGaussianOptimizer

def compute_epsilon(steps=None, delta=1):
  """Computes epsilon value for given hyperparameters."""
  if NOISE_MULTIPLIER == 0.0:
    return float('inf')
  # A representative set of alphas (in the context of Renyi differential privacy (RDP) )
  # on which a RDP privacy guarantee will be calculated to find an approximate optimization
  # of epsilon for a given delta.
  orders = [1 + x / 10. for x in range(1, 100)] + list(range(12, 64))

  sampling_probability = BATCH_SIZE / 100
  rdp = compute_rdp(q=sampling_probability,
                    noise_multiplier=NOISE_MULTIPLIER,
                    steps=steps,
                    orders=orders)

  dp_gaurantee = {
    'eps': get_privacy_spent(orders, rdp, target_delta=delta)[0],
    'delta': delta
  }
  return dp_gaurantee
```

Non-Private Training

```
Current epoch: 0        Stats
        Train accuracy: 0.330
        Test accuracy: 0.365
        Train loss: 1.100
        Test loss: 1.080

Current epoch: 10       Stats
        Train accuracy: 0.873
        Test accuracy: 0.873
        Train loss: 0.668
        Test loss: 0.632

Current epoch: 20       Stats
        Train accuracy: 0.923
        Test accuracy: 0.897
        Train loss: 0.501
        Test loss: 0.466

Current epoch: 30       Stats
        Train accuracy: 0.957
        Test accuracy: 0.941
        Train loss: 0.411
        Test loss: 0.378
```

```
Current epoch: 170      Stats average
        Train accuracy: 0.977
        Test accuracy: 0.961
        Train loss: 0.109
        Test loss: 0.111

Current epoch: 180      Stats average
        Train accuracy: 0.980
        Test accuracy: 0.958
        Train loss: 0.103
        Test loss: 0.111

Current epoch: 190      Stats average
        Train accuracy: 0.987
        Test accuracy: 0.959
        Train loss: 0.101
        Test loss: 0.106

TRAINING COMPLETE.
Current epoch: 199      Model stats:
        Train accuracy: 1.000
        Test accuracy: 0.960
        Train loss: 0.001
        Test loss: 0.102
```

Private Training Strategy

Early stopping thresholds: So as to minimize epsilon for our DP(epsilon,delta) guarantee, we wish to run as few cycles of noisy SGD as possible. To accomplish this, we define thresholds on model performance. Once the model performs as good or better than the thresholds we set, we stop training early. The thresholds set in our code are:

```
TEST_ACCURACY_THRESHOLD = 0.9
TRAIN_ACCURACY_THRESHOLD = 0.9
TEST_LOSS_THRESHOLD = 0.4
TRAIN_LOSS_THRESHOLD = 0.4
```

Choosing Delta: As a general rule of thumb, we want delta to be sizably less than the inverse of the size of our training set. Our training set has 100 examples, so we pick delta to be .001.

Private Training

with NOISE_MULTIPLIER = 5.0

For delta=0.001, we get epsilon 0.65. This result is noted now and discussed in upcoming slides.

Note: Due to the high amount of noise, the model gets better and worse many times before finally getting good enough.

```
Current epoch: 50       Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 0.55
        Train accuracy: 0.640
        Test accuracy: 0.598
        Train loss: 0.722
        Test loss: 0.781

Current epoch: 60       Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 0.60
        Train accuracy: 0.630
        Test accuracy: 0.622
        Train loss: 1.012
        Test loss: 1.034

Current epoch: 70       Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 0.65
        Train accuracy: 0.710
        Test accuracy: 0.694
        Train loss: 0.565
        Test loss: 0.587

Current epoch: 71       Model stats:
        For delta=1.00e-03, the current epsilon is: 0.65
        Train accuracy: 1.000
        Test accuracy: 0.920
        Train loss: 0.138
        Test loss: 0.319
```

Private Training

with NOISE_MULTIPLIER = 2.0

For delta=0.001, we get epsilon 2.94.

Note: Due to a smaller about of noise than in the previous slide. Approaches to good solution last for more epochs, but the path is still bumpy.

```
Current epoch: 160      Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 2.76
        Train accuracy: 0.670
        Test accuracy: 0.660
        Train loss: 0.937
        Test loss: 0.882

Current epoch: 170      Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 2.85
        Train accuracy: 0.670
        Test accuracy: 0.660
        Train loss: 0.668
        Test loss: 0.652

Current epoch: 180      Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 2.94
        Train accuracy: 0.713
        Test accuracy: 0.724
        Train loss: 0.444
        Test loss: 0.410

Current epoch: 181      Model stats:
        For delta=1.00e-03, the current epsilon is: 2.94
        Train accuracy: 1.000
        Test accuracy: 0.920
        Train loss: 0.236
        Test loss: 0.379
```

Private Training
with NOISE_MULTIPLIER = 10

This is too much noise. During training, the model gets very lost and does not find a good fit.

```
Current epoch: 170      Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 0.50
        Train accuracy: 0.340
        Test accuracy: 0.320
        Train loss: 10.638
        Test loss: 10.960

Current epoch: 180      Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 0.51
        Train accuracy: 0.340
        Test accuracy: 0.320
        Train loss: 10.638
        Test loss: 10.960

Current epoch: 190      Stats averaged over last 3 epochs:
        For delta=1.00e-03, the current epsilon is: 0.53
        Train accuracy: 0.340
        Test accuracy: 0.320
        Train loss: 10.638
        Test loss: 10.960

In 200 epochs, the desired thresholds for model accuracy were not achieved.
Current epoch: 199      Model stats:
        For delta=1.00e-03, the current epsilon is: 0.54
        Train accuracy: 0.000
        Test accuracy: 0.320
        Train loss: 16.118
        Test loss: 10.960
```

What this means for Iris

Our best model is (0.001, 0.65) differentially private. Thus, if P is the probability of a bad outcome for Iris.

-- P_iris_participates <= e^[0.65] * P_iris_dosen't_participate + 0.001

--e^[0.65] is about 1.92.

--0.001 is small so let's ignore that term.

Thus in the worst case, P_iris_participates is about 1.92 * P_iris_dosen't_participate.

If Jack learns little about generic Iris classification from our model, then P_iris_dosen't_participate will be small and a doubling of it is tolerable. However, if Jack is good with TensorFlow, than P_iris_dosen't_participate may be large, and Iris would likely want to avoid doubling that risk.

Private Aggregation of Teacher Ensembles (PATE)

1. Partition the original dataset "d" into disjoint sets and train a teacher model on each set.

2. Label a set of unlabeled non-private examples using a noisy voting algorithm "M" (over the teachers) in order to produce a second dataset "c".

3. Train a student model "A" on "c".

PATE Schematic

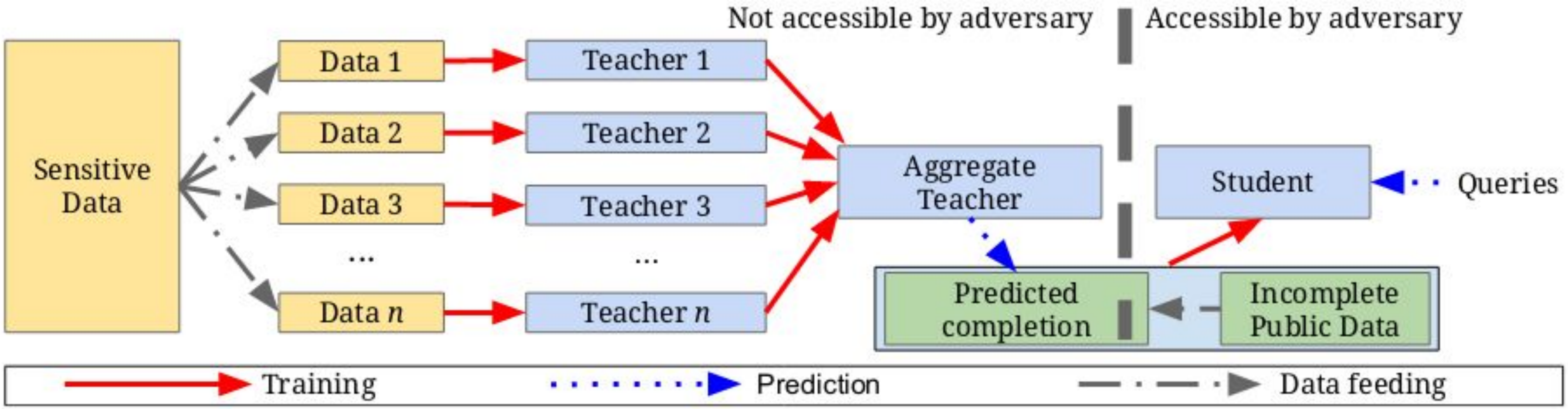


Figure 1: Overview of the approach: (1) an ensemble of teachers is trained on disjoint subsets of the sensitive data, (2) a student model is trained on public data labeled using the ensemble.

Image: Papernot et al., 2017

Performance of noisy-SDG and PATE

We will review experimental performance obtained for noisy-SDG (in Abadi et al, 2016) and for PATE (in Papernot et al, 2017) on the following datasets:

--MNIST (for both noisy SGD and PATE)

--SVHN (for PATE)

--CIFAR-10 (for noisy SGD)

First, a brief discussion of the MNIST, SVHN, and CIFAR-10 datasets...

MNIST

The **MNIST database** (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.
--https://en.wikipedia.org/wiki/MNIST_database

Figure. MnistExamples (by Josef Steppan CC BY-SA 4.0).

SVHN

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.
--http://ufldl.stanford.edu/housenumbers/

Figure: http://ufldl.stanford.edu/housenumbers/

CIFAR-10

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
--https://www.cs.toronto.edu/~kriz/cifar.html

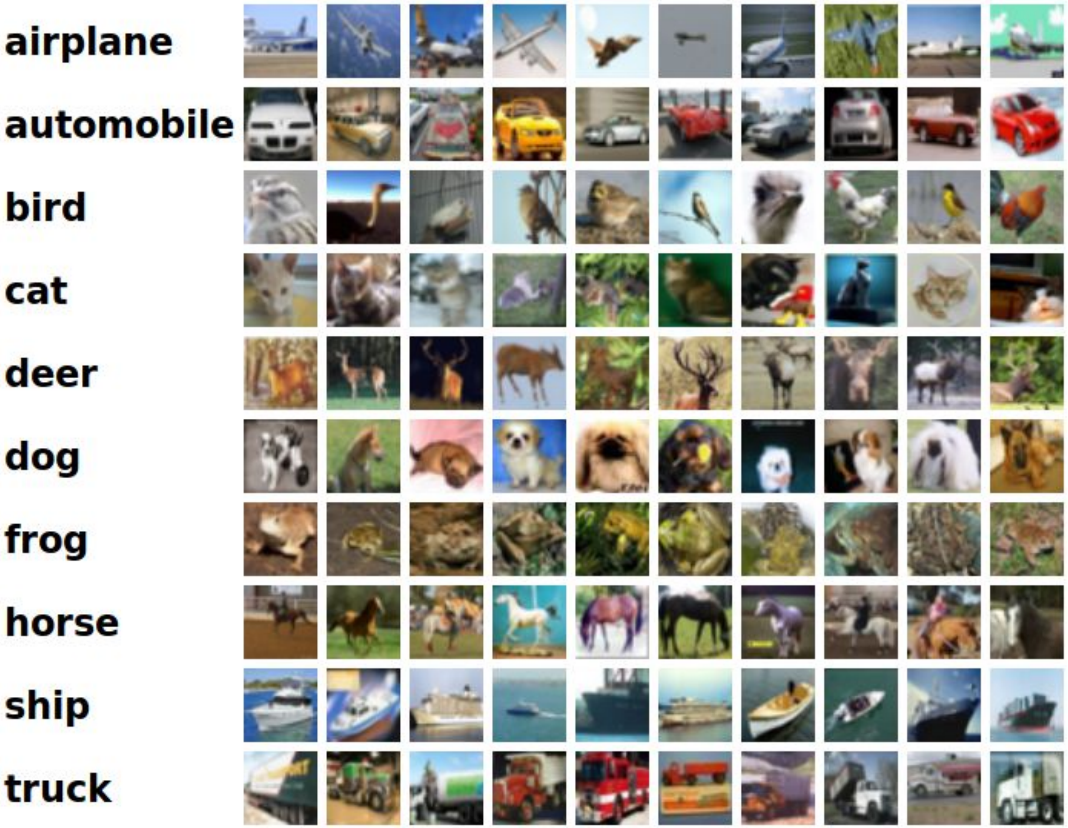


Figure: https://www.cs.toronto.edu/~kriz/cifar.html

Noisy SGD Performance in Abadi et al., 2016

7. CONCLUSIONS

We demonstrate the training of deep neural networks with differential privacy, incurring a modest total privacy loss, computed over entire models with many parameters. In our experiments for MNIST, we achieve 97% training accuracy and for CIFAR-10 we achieve 73% accuracy, both with $(8, 10^{-5})$-differential privacy. Our algorithms are based on a

(Abadi et al., 2016)

PATE Performance in Papernot et al., 2017

Dataset	ε	δ	**Queries**	**Non-Private Baseline**	**Student Accuracy**
MNIST	2.04	10^{-5}	100	99.18%	98.00%
MNIST	8.03	10^{-5}	1000	99.18%	98.10%
SVHN	5.04	10^{-6}	500	92.80%	82.72%
SVHN	8.19	10^{-6}	1000	92.80%	90.66%

Figure 4: **Utility and privacy of the semi-supervised students:** each row is a variant of the student model trained with generative adversarial networks in a semi-supervised way, with a different number of label queries made to the teachers through the noisy aggregation mechanism. The last column reports the accuracy of the student and the second and third column the bound ε and failure probability δ of the (ε, δ) differential privacy guarantee.

(Papernot et al., 2017)