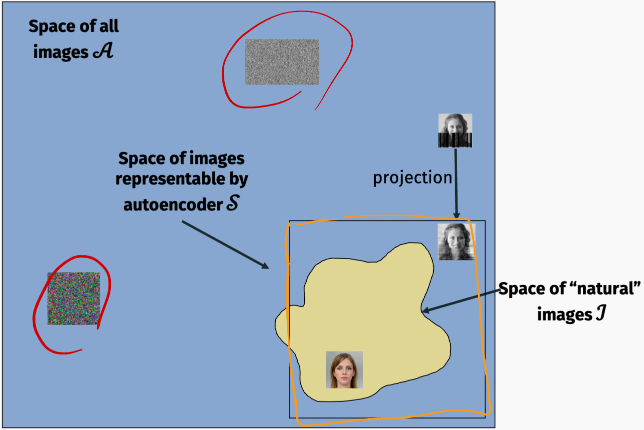
CS-GY 6923: Lecture 14
Image Generation, and Privacy Concerns in ML

NYU Tandon School of Engineering, Akbar Rafiey

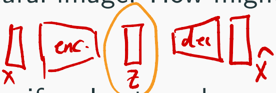
Recap: Autoencoders learn compressed representations



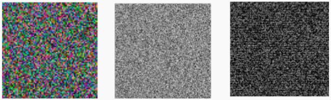
$f(\mathbf{x}) = d(e(\mathbf{x}))$ projects an image $\mathbf{x}$ closer to the space of natural images.

Autoencoders for data generation

Suppose we want to generate a random natural image. How might we do that?



- **Option 1**: Draw each pixel value in $\mathbf{x}$ uniformly at random. Draws a random image from $\mathcal{A}$.

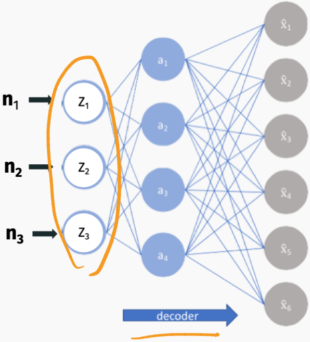


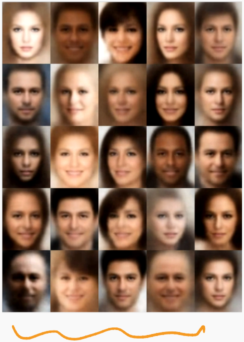
- **Option 2**: Draw $\mathbf{x}$ randomly from $\mathcal{S}$, the space of images representable by the autoencoder.

How do we randomly select an image from $\mathcal{S}$?

Autoencoders for data generation

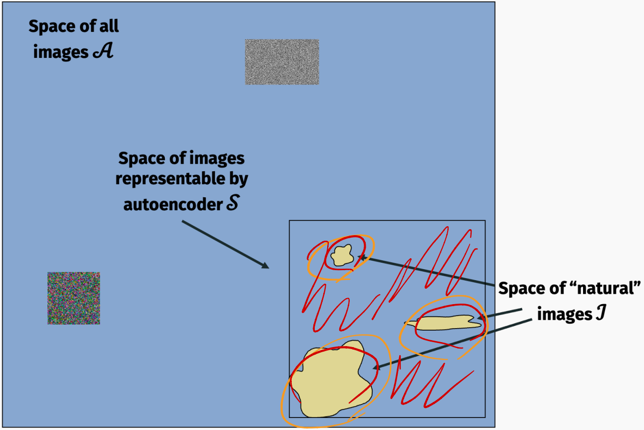
Autoencoder approach to generative ML: Feed random inputs into decoder to produce random realistic outputs.





Main issue: most random inputs will "miss" and produce garbage results.

Autoencoders for data generation



Variational Auto-Encoders (VAEs) attempt to resolve this issue.

Variational AutoEncoders (VAEs)

VAEs attempt to resolve this issue. Basic ideas:

- Instead of mapping inputs to a single latent vector, VAEs map them to a probability distribution in the latent space (e.g., a Gaussian distribution)

smile
skincolor

$N(\mu_1, \sigma_1)$

$N(\mu_2, \sigma_2)$

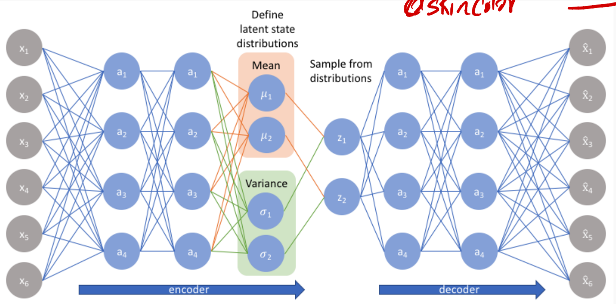


Image from https://www.jeremyjordan.me/variational-autoencoders/

nice dis. ⇒ good images

Variational AutoEncoders (VAEs)

Basic ideas:

- Suppose there exists some hidden variable $\boldsymbol{z}$ which generates $\boldsymbol{x}$.
- Ideally we want to understand $p(\boldsymbol{z} \mid \boldsymbol{x})$ (probabilistic encoder) and $p(\boldsymbol{x} \mid \boldsymbol{z})$ (probabilistic decoder)
- We can only see the data. Computing $p(\boldsymbol{z} \mid \boldsymbol{x})$ is hard

$$p(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})}$$

?

Variational AutoEncoders (VAEs)

Basic ideas:

- Let's approximate $p(\boldsymbol{z} \mid \boldsymbol{x})$ using a simpler to understand distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$.

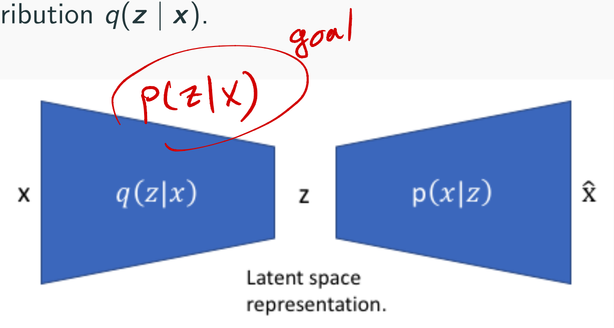


Image from https://www.jeremyjordan.me/variational-autoencoders/

Variational AutoEncoders (VAEs)

Basic ideas:

- Let's approximate $p(\boldsymbol{z} \mid \boldsymbol{x})$ using a simpler to understand distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$.
- $q(\boldsymbol{z} \mid \boldsymbol{x})$ must have nice properties e.g., it should be as similar as possible to $p(\boldsymbol{z} \mid \boldsymbol{x})$
- Optimization problem !

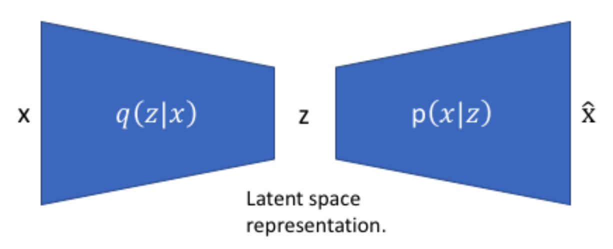


Image from https://www.jeremyjordan.me/variational-autoencoders/

Kullback–Leibler divergence

KL divergence is a measure of difference between two probability distributions.

$$KL(P, Q) = \sum_{x} P(x) \frac{\log(P(x))}{\log(Q(x))}$$

Back to our optimization problem: $q(\boldsymbol{z} \mid \boldsymbol{x})$ and $p(\boldsymbol{z} \mid \boldsymbol{x})$ should be similar

$$\min KL(q(\boldsymbol{z} \mid \boldsymbol{x}), p(\boldsymbol{z} \mid \boldsymbol{x}))$$

Equivalent to:

$$\max \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})} \log(p(\boldsymbol{x} \mid \boldsymbol{z})) - KL(q(\boldsymbol{z} \mid \boldsymbol{x}), p(\boldsymbol{z}))$$

VAE objective

Back to our optimization problem: $q(\boldsymbol{z} \mid \boldsymbol{x})$ and $p(\boldsymbol{z} \mid \boldsymbol{x})$ should be similar

$$\min KL(q(\boldsymbol{z} \mid \boldsymbol{x}), p(\boldsymbol{z} \mid \boldsymbol{x}))$$

Equivalent to:

$$\max \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})} \log(p(\boldsymbol{x} \mid \boldsymbol{z})) - KL(q(\boldsymbol{z} \mid \boldsymbol{x}), p(\boldsymbol{z}))$$

What is the first term? what is the second term?

VAE objective

Back to our optimization problem: $q(\boldsymbol{z} \mid \boldsymbol{x})$ and $p(\boldsymbol{z} \mid \boldsymbol{x})$ should be similar

$$\min KL(q(\boldsymbol{z} \mid \boldsymbol{x}), p(\boldsymbol{z} \mid \boldsymbol{x}))$$

Equivalent to:

$$\max \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})} \log(p(\boldsymbol{x} \mid \boldsymbol{z})) - KL(q(\boldsymbol{z} \mid \boldsymbol{x}), p(\boldsymbol{z}))$$

First term: reconstruction likelihood.

Second term: ensures that our learned distribution q is similar to the true prior distribution p.

VAEs: implementation

Have neural networks to learn the mappings $q(\boldsymbol{z} \mid \boldsymbol{x})$ and $p(\boldsymbol{x} \mid \boldsymbol{z})$.

$$\max_{\theta,\phi} \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log(p_\theta(\boldsymbol{x} \mid \boldsymbol{z})) - KL(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}), p_\theta(\boldsymbol{z}))$$

Assumptions: $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$, $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$, and $p_\theta(\boldsymbol{z})$ are Gaussian distributions.

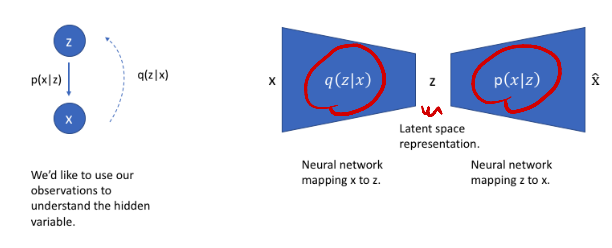


Image from https://www.jeremyjordan.me/variational-autoencoders/

$\phi = [w_1, w_2, w_3 \ldots]$ parameters of encoder NN. that learns $q_\phi(z|x)$

VAEs: implementation

The encoder model of a VAE will output parameters describing a distribution for each dimension in the latent space.

Assuming Gaussian we only need a mean and a variance for describing each dimension in the latent space

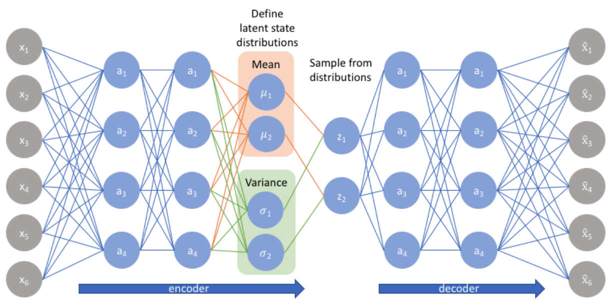


Image from https://www.jeremyjordan.me/variational-autoencoders/

VAEs: implementation

It is not easy to backpropagate the gradient through samples !

Reparameterization technique addresses this issue.

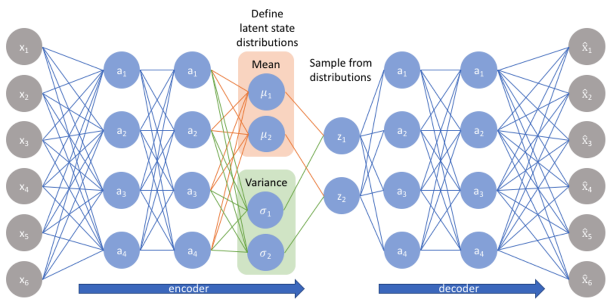


Image from https://www.jeremyjordan.me/variational-autoencoders/

VAEs: reparameterization technique

$\boldsymbol{z} \sim q_\phi(\cdot \mid \boldsymbol{x})$ is normally distributed, as

$$\mathcal{N}(\mu_\phi(\boldsymbol{x}), \sigma_\phi(\boldsymbol{x}))$$

This can be reparameterized by letting $\varepsilon \sim \mathcal{N}(0, I)$, and constructing $\boldsymbol{z}$ as

$$\boldsymbol{z} = \mu_\phi(\boldsymbol{x}) + L_\phi(\boldsymbol{x})\varepsilon$$

Here, $\sigma_\phi(\boldsymbol{x})$ is obtained by the Cholesky decomposition:

$$\sigma_\phi(\boldsymbol{x}) = L_\phi(\boldsymbol{x})L_\phi(\boldsymbol{x})^T$$

Then we have

$$\nabla_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\cdot|\boldsymbol{x})}\left[\ln \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] = \mathbb{E}_\varepsilon\left[\nabla_\phi \ln \frac{p_\theta(\boldsymbol{x}, \mu_\phi(\boldsymbol{x}) + L_\phi(\boldsymbol{x})\varepsilon)}{q_\phi(\mu_\phi(\boldsymbol{x}) + L_\phi(\boldsymbol{x})\varepsilon|\boldsymbol{x})}\right]$$

and so we obtain an unbiased estimator of the gradient, allowing stochastic gradient descent.

VAEs: reparameterization technique

Since we reparameterized z, we need to find $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Let q_0 be the probability density function for ε, then

$$\ln q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \ln q_0(\varepsilon) - \ln \left|\det\left(J(\boldsymbol{z}, \varepsilon)\right)\right|$$

Since $\boldsymbol{z} = \mu_\phi(\boldsymbol{x}) + L_\phi(\boldsymbol{x})\varepsilon$, this becomes

$$\ln q_\phi(\boldsymbol{z}|\boldsymbol{x}) = -\frac{1}{2}\|\varepsilon\|^2 - \ln|\det L_\phi(\boldsymbol{x})| - \frac{n}{2}\ln(2\pi)$$

VAEs: reparameterization technique

We can now optimize the parameters of the distribution (unbiased estimation of the gradient) while still maintaining the ability to randomly sample from that distribution.

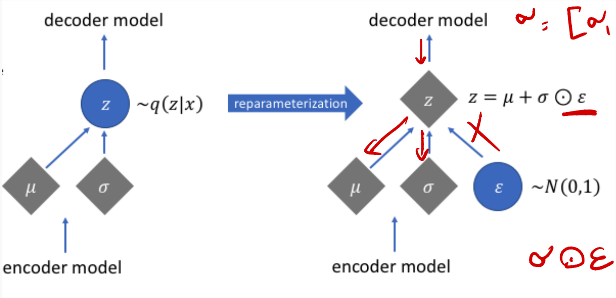
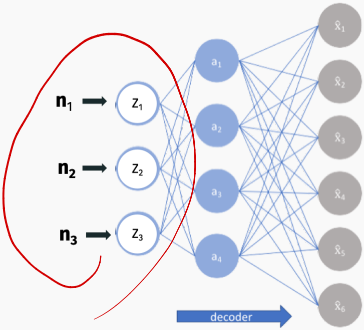
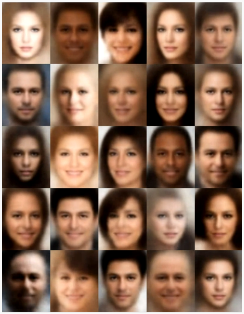


Image from https://www.jeremyjordan.me/variational-autoencoders/

Generative Adversarial Networks (GANs)

VAEs give very good results, but tends to produce images with immediately recognizable flaws (e.g. soft edges, high-frequency artifacts).





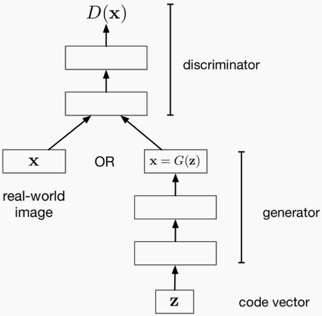
Generative Adversarial Networks (GANs)

Lots of efforts to hand-design regularizers that penalize images that don't look realistic to the human eye.

Main idea behind GANs: Use machine learning to automatically encourage realistic looking images.

$$\min_{\theta} L(\theta) + P(\theta)$$

Generative Adversarial Networks (GANs)



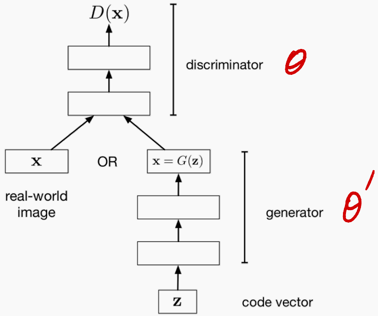
Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be real images and let $\mathbf{z}_1, \ldots, \mathbf{z}_m$ be random code vectors. The goal of the discriminator is to output a number between $[0, 1]$ which is close to 0 if the image is fake, close to 1 if it's real.

Train weights of discriminator $D_{\boldsymbol{\theta}}$ to minimize:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} -\log\left(D_{\boldsymbol{\theta}}(\mathbf{x}_i)\right) + \sum_{i=1}^{m} -\log\left(1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\theta}'}(\mathbf{z}_i))\right)$$

0.99

Generative Adversarial Networks (GANs)



Goal of the generator $G_{\theta'}$ is the opposite. We want to maximize:

$$\max_{\theta'} \sum_{i=1}^{m} -\log\left(1 - D_{\theta}(G_{\theta'}(\mathbf{z}_i))\right)$$

This is called an "adversarial loss function". ~~D~~ G is playing the role of the adversary.

Generative Adversarial Networks (GANs)

$$\boldsymbol{\theta}^*, \boldsymbol{\theta}'^* \text{ solve } \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\theta}'} \sum_{i=1}^{n} -\log\left(D_{\boldsymbol{\theta}}(\mathbf{x}_i)\right) + \sum_{i=1}^{m} -\log\left(1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\theta}'}(\mathbf{z}_i)\right)$$

This is called a minimax optimization problem. Really tricky to solve in practice.

- **Repeatedly play:** Fix one of $\boldsymbol{\theta}^*$ or $\boldsymbol{\theta}'^*$, train the other to convergence, repeat.
- **Simultaneous gradient descent:** Run a single gradient descent step for each of $\boldsymbol{\theta}^*, \boldsymbol{\theta}'^*$ and update D and G accordingly. Difficult to balance learning rates.
- Lots of tricks (e.g. slight different loss functions) can help.

Generative Adversarial Networks (GANs)

State of the art until a few years ago.

Quality of generative model

How to evaluate the quality of our model e.g., GAN? What do we expect from it?

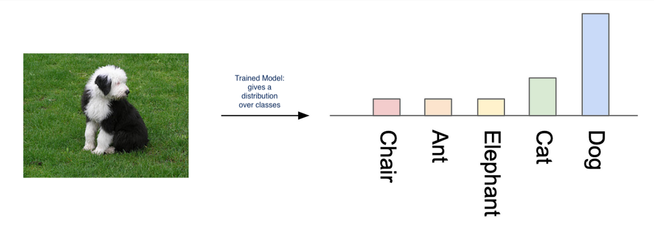
Quality of generative model

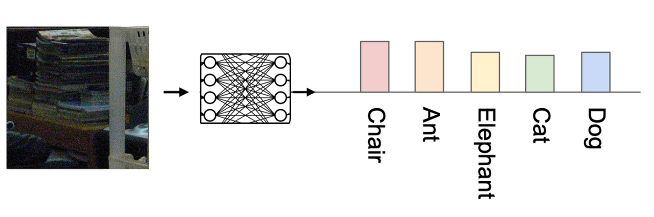
How to evaluate the quality of our model e.g., GAN? What do we expect from it?

- The images generated by our model should have variety (e.g., each image is a different breed of dog)
- Each image distinctly looks like something (e.g., one image is clearly a Poodle, the next a great example of a French Bulldog)
- ...

The Inception Score (IS)

Each image distinctly looks like something (e.g., one image is clearly a Poodle, the next a great example of a French Bulldog)

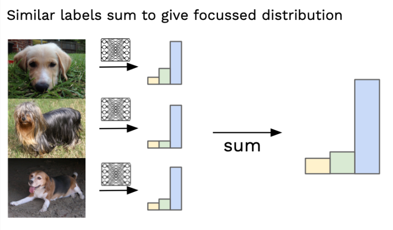


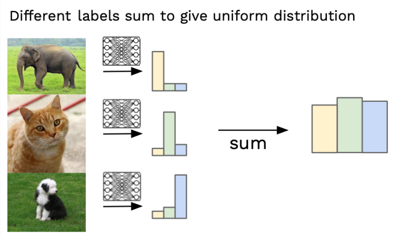


The Inception Score (IS)

The images generated by our model should have variety:

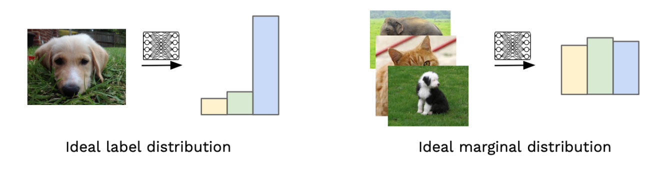
Generate a lot of images (50,000) using the model and sum their distributions.





The Inception Score (IS)

Higher KL divergence, means better score.



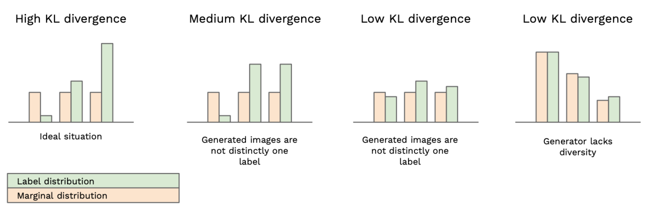


Image from
https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a

Diffusion Model

Diffusion

Auto-encoder/VAE, GAN approach: Input noise, map directly to image and vice versa.

Diffusion: Slowly move from noise to image and vice versa.

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

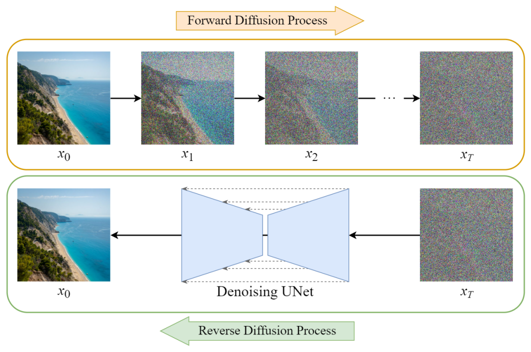
Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

Abstract

We present high quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics. Our best results are obtained by training on a weighted variational bound designed according to a novel connection between diffusion probabilistic

How diffusion models work

- Forward Process:
  - Gradually add noise to data until it becomes pure noise.
- Reverse Process:
  - Train a neural network to remove the noise step by step.

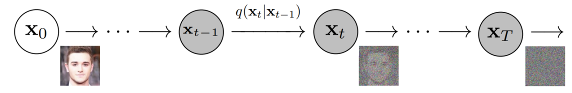


Key Question: How do we predict and reverse noise effectively?

Mathematical Formulation (1/2)

Forward Process (Adding Noise):

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$$



- β_t: Noise schedule.
- After T steps, for large enough T, $\mathbf{x}_T$ is pure noise.

Cumulative Noise:

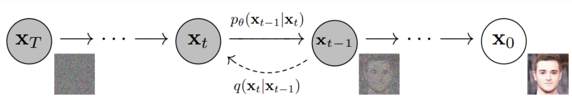
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

with retention factor $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$.

Mathematical formulation (2/2)

Reverse Process (Denoising):

$$p_\theta(\mathbf{x}_{t-1}|x_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$



- μ_θ: Predicted mean of the clean image.
- Σ_θ: Predicted variance (optional).

Training objective:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0, t, \epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2\right]$$

Diffusion models vs VAEs

- Recall the goal of VAE was to have a probabilistic representation of latent space attributes. This was done in one shot, from image to Gaussian distributions.
- VAE decoder does the reverse in one shot. Takes Gaussian noise and returns an image.
- Diffusion model doing more or less the same but with many careful intermediate noising and denoising steps.
- There are fundamental differences ...

Diffusion process

- A diffusion process is a stochastic Markov process having continuous path
- stochastic Markov process: future state will only depend on the current state, knowing the past does not change anything
- continuous: no jumps
- Allows to transition from complex distributions to simple distributions

Forward process: a closer look

Forward process: $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$

Why does it converge to a simple distribution ?

Suppose the process is $\mathbf{x}_t = \alpha \mathbf{x}_{t-1} + \beta \mathcal{N}(0, I)$.

What values of α and β makes sense for the process?

- $\alpha = 0, \beta = 1$?
- $\alpha \geq 1, \beta \leq 1$?
- Seems something like $\alpha = \sqrt{0.99}, \beta = \sqrt{0.01}$ is appropriate.

Forward process: a closer look

Let's check if the following α and β converges:

$$\mathbf{x}_t = \sqrt{1-\beta}\mathbf{x}_{t-1} + \sqrt{\beta}\mathcal{N}(0, I)$$

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{1-\beta}\mathbf{x}_{t-1} + \sqrt{\beta}\mathcal{N}(0, I) \\
&= \sqrt{1-\beta}\left(\sqrt{1-\beta}\mathbf{x}_{t-2} + \sqrt{\beta}\mathcal{N}(0, I)\right) + \sqrt{\beta}\mathcal{N}(0, I) \\
&= (\sqrt{1-\beta})^2\mathbf{x}_{t-2} + \sqrt{1-\beta}\sqrt{\beta}\mathcal{N}(0, I) + \sqrt{\beta}\mathcal{N}(0, I)
\end{aligned}$$

Forward process: a closer look

Let's check if the following α and β converges:

$$\mathbf{x}_t = \sqrt{1-\beta}\mathbf{x}_{t-1} + \sqrt{\beta}\mathcal{N}(0, I)$$

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{1-\beta}\mathbf{x}_{t-1} + \sqrt{\beta}\mathcal{N}(0, I) \\
&= \sqrt{1-\beta}(\sqrt{1-\beta}\mathbf{x}_{t-2} + \sqrt{\beta}\mathcal{N}(0, I)) + \sqrt{\beta}\mathcal{N}(0, I) \\
&= (\sqrt{1-\beta})^2\mathbf{x}_{t-2} + \sqrt{1-\beta}\sqrt{\beta}\mathcal{N}(0, I) + \sqrt{\beta}\mathcal{N}(0, I) \\
&\dots \\
&= (\sqrt{1-\beta})^t\underbrace{\mathbf{x}_{t-t}}_{x_0} + \dots + (\sqrt{1-\beta})^2\sqrt{\beta}\mathcal{N}(0, I) + \sqrt{1-\beta}\sqrt{\beta}\mathcal{N}(0, I) \\
&+ \sqrt{\beta}\mathcal{N}(0, I)
\end{aligned}$$

Forward process: a closer look

$$\mathbf{x}_t = (\sqrt{1-\beta})^t \mathbf{x}_0 + \cdots + (\sqrt{1-\beta})^2 \sqrt{\beta}\mathcal{N}(0, I) + \sqrt{1-\beta}\sqrt{\beta}\mathcal{N}(0, I) + \sqrt{\beta}\mathcal{N}(0, I)$$

0

- for large t, $(\sqrt{1-\beta})^t$ approaches to 0
- each of $(\sqrt{1-\beta})^i \sqrt{\beta}\mathcal{N}(0, I)$ is a Gaussian with mean 0 and variance $(1-\beta)^i \beta$.
- All these Gaussian can be written as one Gaussian distribution with variance:

$$\sum_{i=0}^{t} (1-\beta)^i \beta = \beta \frac{1-(1-\beta)^t}{1-(1-\beta)} \sim \frac{\beta}{\beta} = 1$$

For large enough t we converge to the Normal distribution $\mathcal{N}(0, I)$.

Forward process: a closer look

at any time step
we know β_t.

- In practice we do not use a fixed noise β
- Linear schedule noise: $\beta_1 = 0.0001$ and $\beta_T = 0.02$
- The number of steps is about 10000 (application dependent), but this is slow!
- Recall $\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\mathcal{N}(0, I)$.
- Let's define $\alpha_t = 1 - \beta_t$ and do the recursive expansion.

Forward process: a closer look

$\alpha_t = 1 - \beta_t$

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\mathcal{N}(0, I) \\
&= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\mathcal{N}(0, I) \\
&= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\mathcal{N}(0, I)) + \sqrt{1-\alpha_t}\mathcal{N}(0, I) \\
&= (\sqrt{\alpha_t\alpha_{t-1}})\mathbf{x}_{t-2} + \sqrt{1-\alpha_t+\alpha_t-\alpha_t\alpha_{t-1}}\sqrt{\beta}\mathcal{N}(0, I) \\
&\dots
\end{aligned}
$$

$1 - \alpha_t\alpha_{t-1}$

Forward process: a closer look

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\mathcal{N}(0, I) \\
&= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\mathcal{N}(0, I) \\
&= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\mathcal{N}(0, I)) + \sqrt{1-\alpha_t}\mathcal{N}(0, I) \\
&= (\sqrt{\alpha_t\alpha_{t-1}})\mathbf{x}_{t-2} + \sqrt{1-\alpha_t+\alpha_t-\alpha_t\alpha_{t-1}}\sqrt{\beta}\mathcal{N}(0, I) \\
&\cdots \\
&= \sqrt{\alpha_t\alpha_{t-1}\cdots\alpha_2\alpha_1}\mathbf{x}_0 + \sqrt{1-\alpha_t\alpha_{t-1}\cdots\alpha_2\alpha_1}\mathcal{N}(0, I) \\
&= \sqrt{\prod_{i=1}^{t}\alpha_i\mathbf{x}_0 + \sqrt{1-\prod_{i=1}^{t}\alpha_i\mathcal{N}(0, I)}} \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\mathcal{N}(0, I) \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon
\end{aligned}
$$

$$\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

Reverse process: a closer look

Computing the reverse path (reverse denoising distribution) is not, but we know it is diffusion process.

We train a model to approximate the reverse distribution.

Similar to VAEs, the goal is to maximize a lower bound for the likelihood:

$$\log p(\mathbf{x}_0) \geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right]$$

A lot of effort goes into simplify this lower bound and exploiting the fact that we are dealing with diffusion process.

Reverse process: a closer look

At the end of the day, the model should learn the noise added

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0, t, \epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2\right]$$

predicted noise by NN.

actual noise

What does it mean? How do we actually do training ?

Reverse process: training

- Sample an image $\mathbf{x}_0$ and a time step t
- Sample a random noise ε. from $N(0, I)$
- Get the noise image at time t: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon$
- Feed $\mathbf{x}_t$ to the neural network.
- The model's predicted noise ε_θ should be close to ε.

Algorithm 1 Training

1: **repeat**
2: $\quad \mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $\quad t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: $\quad$ Take gradient descent step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$

6: **until** converged

noise predicted by my NN_θ.

Reverse process: image generation

- Start from a noise image $\mathbf{x}_T \sim \mathcal{N}(0, I)$.
- Feed this to the trained neural network to predict a noise $\varepsilon_\theta(\mathbf{x}_T)$
- Given this predicted noise we can do denoising and obtain $\mathbf{x}_{T-1}$ (we skipped through the math here)
- Repeat the above until we get to $\mathbf{x}_0$.

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

we can generate x_{t-1}:

having predicted noise

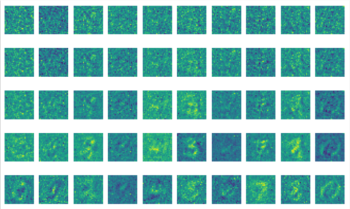
Semantic embeddings + diffusion

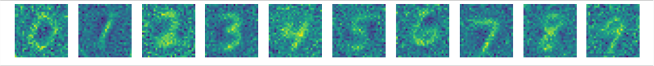
Text to image synthesis: Dall-E, Imagen, Stable Diffusion

"A chair that looks like a pineapple"

Diffusion

A demo for generating digits by training on MNIST.





Ethical challenges
How to preserve privacy?

Generative models and data leakage

Generative models can potentially memorise and regenerate their training data points.

Extracting Training Data from Diffusion Models

Nicholas Carlini[*1] *Jamie Hayes*[*2] *Milad Nasr*[*1]
Matthew Jagielski[+1] *Vikash Sehwag*[+4] *Florian Tramèr*[+3]
Borja Balle[†2] *Daphne Ippolito*[†1] *Eric Wallace*[†5]
[1]Google [2]DeepMind [3]ETHZ [4]Princeton [5]UC Berkeley
[*]Equal contribution [+]Equal contribution [†]Equal contribution

Generative models and data leakage

Generative models can potentially memorise and regenerate their training data points.

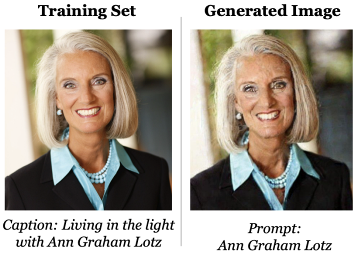


Figure 1: Diffusion models memorize individual training examples and generate them at test time. **Left:** an image from Stable Diffusion's training set (licensed CC BY-SA 3.0, see [49]). **Right:** a Stable Diffusion generation when prompted with "Ann Graham Lotz". The reconstruction is nearly identical (ℓ_2 distance = 0.031).

Data leakage

As we saw in the text generation lab, machine learning algorithms are prone to leak information about their training data:

> arm towards the viewer. Gregor then turned to look out the window at a barren
> sister only needed to hear the visitor's first words of greeting and he knew who
> calm, "I'll get dressed straight away now, pack up my samples and set off. Will
> again, "seven o'clock, and there's still a fog like this." And he lay there sighing,
> harder than before, if that was possible, he felt that the lower part of his body a

Here, our generative model revealed entire sentences from the training input. This is a quality issue, but can also be a privacy issue.

Data leakage

Many modern ML systems trained on user data.

- Smart Compose in Gmail (trained on user emails).
- Generative AI for medical record taking (trained on patient health data).
- Github Copilot trained on public and private repositories.

Even if models do not directly generate private data, it can sometimes be extracted from them.

Data leakage

Training data extraction attacks can reconstruct verbatim training examples e.g., they can extract secrets such as verbatim social security numbers or passwords.

Extracting Training Data from Large Language Models

Nicholas Carlini[1] Florian Tramèr[2] Eric Wallace[3] Matthew Jagielski[4]
Ariel Herbert-Voss[5,6] Katherine Lee[1] Adam Roberts[1] Tom Brown[5]
Dawn Song[3] Úlfar Erlingsson[7] Alina Oprea[4] Colin Raffel[1]
[1]*Google* [2]*Stanford* [3]*UC Berkeley* [4]*Northeastern University* [5]*OpenAI* [6]*Harvard* [7]*Apple*

Abstract

It has become common to publish large (billion parameter) language models that have been trained on private datasets. This paper demonstrates that in such settings, an adversary can perform a *training data extraction attack* to recover individual training examples by querying the language model.

We demonstrate our attack on GPT-2, a language model trained on scrapes of the public Internet, and are able to extract hundreds of verbatim text sequences from the model's training data. These extracted examples include (public) personally identifiable information (names, phone numbers, and email addresses), IRC conversations, code, and 128-bit UUIDs. Our attack is possible even though each of the above sequences are included in just *one* document in the training data.

We comprehensively evaluate our extraction attack to understand the factors that contribute to its success. Worryingly, we find that larger models are more vulnerable than smaller models. We conclude by drawing lessons and discussing possible safeguards for training large language models.

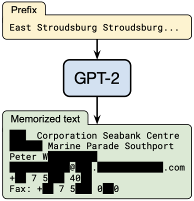


Figure 1: **Our extraction attack.** Given query access to a neural network language model, we extract an individual person's name, email address, phone number, fax number, and physical address. The example in this figure shows information that is all accurate so we redact it to protect privacy.

1 Introduction

The privacy challenge

How do we balance privacy concerns with the desire to train models on as much data as possible?

Formalizing privacy

There have been many many attempts to formalize what it means for a machine learning algorithm or system to be private.

Calibrating Noise to Sensitivity in Private Data Analysis

Cynthia Dwork[1], Frank McSherry[1], Kobbi Nissim[2], and Adam Smith[3*]

[1] Microsoft Research, Silicon Valley. {dwork,mcsherry}@microsoft.com
[2] Ben-Gurion University. kobbi@cs.bgu.ac.il
[3] Weizmann Institute of Science. adam.smith@weizmann.ac.il

Differential Privacy has become the gold standard definition.

Clear theoretical founding, widely used in implemented systems (TensorFlow, US Census statistics, Apple User data, etc.)

Differential privacy

Definition based on notation of neighboring datasets.

Definition: A dataset $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ is neighbors of a dataset $\mathbf{X}' = [\mathbf{x}'_1, \ldots, \mathbf{x}'_n]$ if:

$$\mathbf{x}_i = \mathbf{x}'_i \text{ for all but one value of } i \in \{1, \ldots n\}.$$

I.e., $\mathbf{x}_j \neq \mathbf{x}'_j$ for a single index j.

Alternative but closely related definition: $\mathbf{X}$ and $\mathbf{X}'$ are neighbors if $\mathbf{X}'$ can be obtained by adding or removing a single data point from $\mathbf{X}$.

Differential privacy

Definition

An algorithm $\mathcal{A}$ satisfies ϵ-differential privacy if, for any two neighboring datasets $\mathbf{X}$, $\mathbf{X}'$, and any possible output of the algorithm $\mathbf{z}$,

$$\Pr[\mathcal{A}(\mathbf{X}) = \mathbf{z}] \leq e^{\epsilon} \Pr[\mathcal{A}(\mathbf{X}') = \mathbf{z}].$$

In the context of machine learning, $\mathcal{A}$ could be the training procedure and $\mathbf{z}$ could be, e.g., the model weights.

In the context of databases/statistical applications, $\mathcal{A}$ might implement a simple statistic function like the mean:

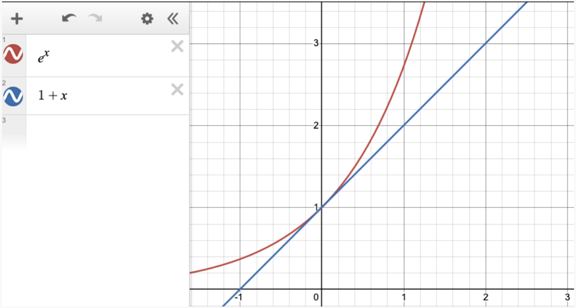
$$\frac{1}{n}\sum_{i=1}^{n} x_i.$$

Differential privacy

Definition

An algorithm $\mathcal{A}$ satisfies ϵ-differential privacy if, for any two neighboring datasets $\mathbf{X}$, $\mathbf{X}'$, and any possible output of the algorithm $\mathbf{z}$, $\Pr[\mathcal{A}(\mathbf{X}) = \mathbf{z}] \leq e^{\epsilon} \Pr[\mathcal{A}(\mathbf{X}') = \mathbf{z}]$.

Think of ϵ as a reasonably small constant. E.g. $\epsilon \in (0, 5]$. For small ϵ, $e^{\epsilon} \approx (1 + \epsilon)$.



Differential privacy

Definition

An algorithm $\mathcal{A}$ satisfies ϵ-differential privacy if, for any two neighboring datasets $\mathbf{X}$, $\mathbf{X}'$, and any possible output of the algorithm $\mathbf{z}$, $\Pr[\mathcal{A}(\mathbf{X}) = \mathbf{z}] \leq e^{\epsilon} \Pr[\mathcal{A}(\mathbf{X}') = \mathbf{z}]$.

In words, differential privacy says that including an individuals data in a dataset $\mathbf{X}$ can only increase or decrease the probability of observing any particular output by a small factor.

Inherently a property of randomized algorithms. Obtaining differentially private machine learning methods will require **adding randomness to the training process**.

Differential privacy properties

Postprocessing property: If an algorithm $\mathcal{A}(\mathbf{X})$ is ϵ-DP, then $\mathcal{B}(\mathcal{A}(\mathbf{X}))$ is ϵ-DP for any (possibly non-private) algorithm $\mathcal{B}$.

Composition property: If an algorithm $\mathcal{A}_1$ is ϵ_1-DP and $\mathcal{A}_2$ is ϵ_2-DP, then $\mathcal{B}(\mathcal{A}_1(\mathbf{X}), \mathcal{A}_2(\mathbf{X}))$ is $(\epsilon_1 + \epsilon_2)$-DP.

Differential privacy

There are many ways to add randomness. Perhaps the most common is noise injection.

Simple example: Suppose $\mathbf{X}$ contains scalar values $x_1, \ldots, x_n \in \{0, 1\}$. Suppose we want to compute the average, $Q(\mathbf{X}) = \frac{1}{n} \sum_{i=1}^{n} x_i$.

Naively, adding or removing a point from the dataset changes the average by $\pm \frac{1}{n}$ with probability 1, so, naively, a mean computation is not differentially private.

Noise injection

Differentially Private Estimate of $Q(\mathbf{X}) = \frac{1}{n}\sum_{i=1}^{n} x_i$:

- Generate an appropriate random number η.
- Return $Q(\mathbf{X}) + \eta$.

Example $= \mathbf{X} = \{0, 1, 1, 0, 0, 0\}, \mathbf{X}' = \{0, 1, 1, 0, 1, 0\}$.

Trade-off between <u>privacy</u> and <u>accuracy</u>.

What type of noise and how much?

Theorem (Laplace Mechanism)

For a function Q with <u>sensitivity</u> Δ_Q,

$$\mathcal{A}(\mathbf{X}) = Q(\mathbf{X}) + Lap(\Delta_Q/\epsilon)$$

is ϵ*-differentially private.*

Sensitiviy $\Delta_Q = \max_{\text{neighboring } \mathbf{X},\mathbf{X}'} |Q(\mathbf{X}) - Q(\mathbf{X}')|$.

What is Δ_Q for $Q(\mathbf{X}) = \frac{1}{n}\sum_{i=1}^{n} x_i$?

$Lap(b)$ is a Laplacian random variable with parameter b (which means variance $2b^2$). PDF is:

$$p_b(\eta) = \frac{1}{2b} e^{-|\eta|/b}$$

Laplace mechanism analysis

Theorem (Laplace Mechanism)

For a function Q with <u>sensitivity</u> Δ_Q,
$\mathcal{A}(\mathbf{X}) = Q(\mathbf{X}) + Lap(\Delta_Q/\epsilon)$ *is* ϵ*-differentially private.*

Proof: For any possible output z,

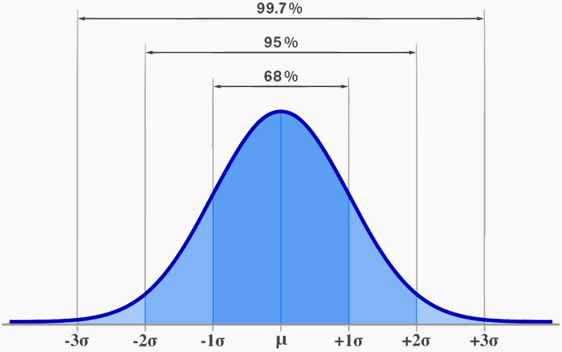
- $\Pr[\mathcal{A}(\mathbf{X}) = z] = \frac{1}{2(\Delta_Q/\epsilon)} e^{-|Q(\mathbf{X})-z|/(\Delta_Q/\epsilon)}$
- $\Pr[\mathcal{A}(\mathbf{X}') = z] = \frac{1}{2(\Delta_Q/\epsilon)} e^{-|Q(\mathbf{X}')-z|/(\Delta_Q/\epsilon)}$

$$\frac{\Pr[\mathcal{A}(\mathbf{X}) = z]}{\Pr[\mathcal{A}(\mathbf{X}') = z]} = e^{-(|Q(\mathbf{X})-z|-|Q(\mathbf{X}')-z|)/(\Delta_Q/\epsilon)}$$

$$\leq e^{\frac{|Q(\mathbf{X})-Q(\mathbf{X}')|}{\Delta_Q/\epsilon}} \leq e^{\epsilon}.$$

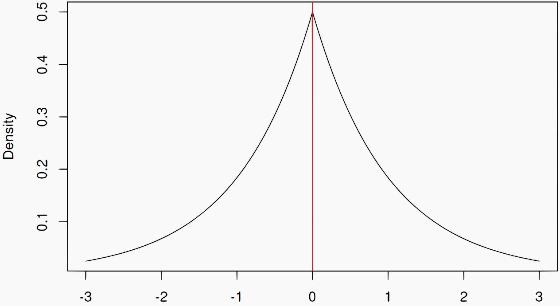
What do we pay in terms of accuracy?

$Lap(b)$ has standard deviation $\sqrt{2}b$. Like Gaussian distribution, Laplace random variables usually fall within a few standard deviations of the mean:



What do we pay in terms of accuracy?

$Lap(b)$ has standard deviation $\sqrt{2}b$. Like Gaussian distribution, Laplace random variables usually fall within a few standard deviations of the mean:



What do we pay in terms of accuracy?

Standard deviation $= \sqrt{2} \cdot \frac{\Delta_Q}{\epsilon}$.

For $x_1, \ldots, x_n \in [0,1]$, $Q(\mathbf{X}) = \frac{1}{n}\sum_{i=1}^{n} x_i$, we have that:

$$\Delta_Q = \frac{1}{n}.$$

Overall error from adding noise:

$$O\left(\frac{1}{\epsilon n}\right)$$

Very reasonable if n is large!

E.g., if $n = 10,000$ can get error roughly .001 on mean estimate with privacy parameter $\epsilon = .1$.

What about more complex functions?

In machine learning applications, Q is an entire training procedure, and the output is vector of parameters.

$$Q(\mathbf{X}, \mathbf{y}) \rightarrow \boldsymbol{\beta} \in \mathbb{R}^d.$$

Challenges:

- Very hard to estimate the sensitivity to figure out how much noise should be added.
- If some parameters are more sensitive to noise, we could change the models output drastically.

Differentially private (stochastic) gradient descent

Main idea: Typically $Q(\mathbf{X}, \mathbf{y})$ is computed by running gradient descent on a loss function $L(\beta)$. Instead of directly adding noise to $Q(\mathbf{X}, \mathbf{y})$, add noise at each step of gradient descent.

Basic Gradient descent algorithm:

- Choose starting point $\beta^{(0)}$.
- For $i = 0, \ldots, T$:
    - $\beta^{(i+1)} = \beta^{(i)} - \eta \nabla L(\beta^{(i)})$
- Return $\beta^{(T)}$.

Differentially private (stochastic) gradient descent

Typical loss function in machine learning have finite sum structure.

$$L(\boldsymbol{\beta}) = \sum_{j=1}^{n} \ell(\boldsymbol{\beta}, \mathbf{x}_j, y_j)$$

By linearity:

$$\nabla L(\boldsymbol{\beta}) = \sum_{j=1}^{n} \nabla \ell(\boldsymbol{\beta}, \mathbf{x}_j, y_j)$$

Looks just like our mean estimation problem! Can bound the contribution of each data example $(\mathbf{x}_j, y_j)$ to the gradient to get a sensitivity, then add noise.

Differentially private (stochastic) gradient descent

Due to a 2016 paper by Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, Li Zhang.

DP-SGD:

- Choose starting point $\beta^{(0)}$.
- For $i = 0, \ldots, T$:
  - $\beta^{(i+1)} = \beta^{(i)} - \eta(\nabla L(\beta^{(i)}) + \mathbf{r}_i)$
- Return $\beta^{(T)}$.

Above each $\mathbf{r}_i$ is a random Gaussian vector.

Leading way to incorperate privacy into training machine learning models. Implented natively, e.g., in TensorFlow.