CS-GY 6923: Lecture 13
Word Embeddings, Image Generation, Ethical AI

NYU Tandon School of Engineering, Akbar Rafiey
Slides by Prof. Christopher Musco

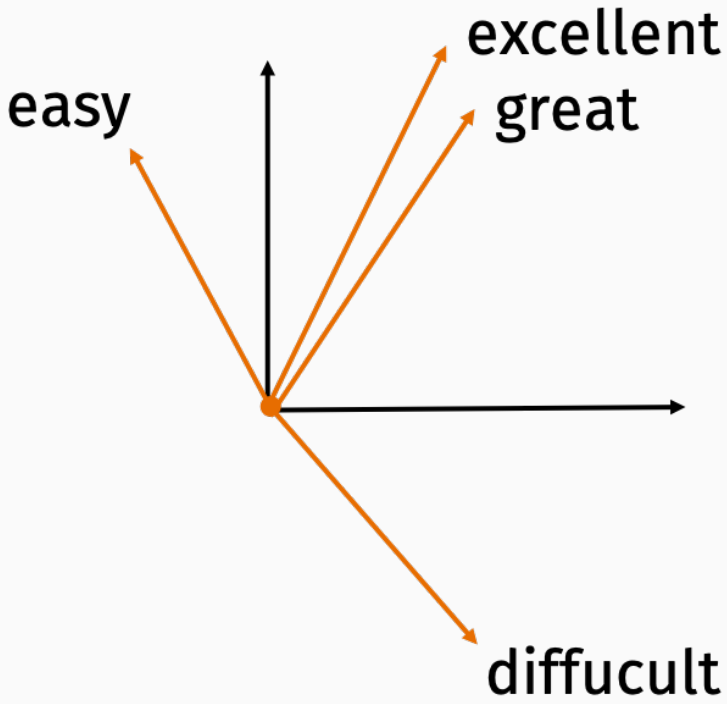
Semantic embeddings

Goal: Learn mapping from numerical inputs into vectors such that similar inputs map to similar vectors (e.g., with high inner product).

Example

Word embeddings

Example: $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ is larger if $word_i$ and $word_j$ appear in more documents together (high value in **word-word co-occurrence matrix**, $\mathbf{X}^T\mathbf{X}$). Similarity of word embeddings mirrors similarity of word context.



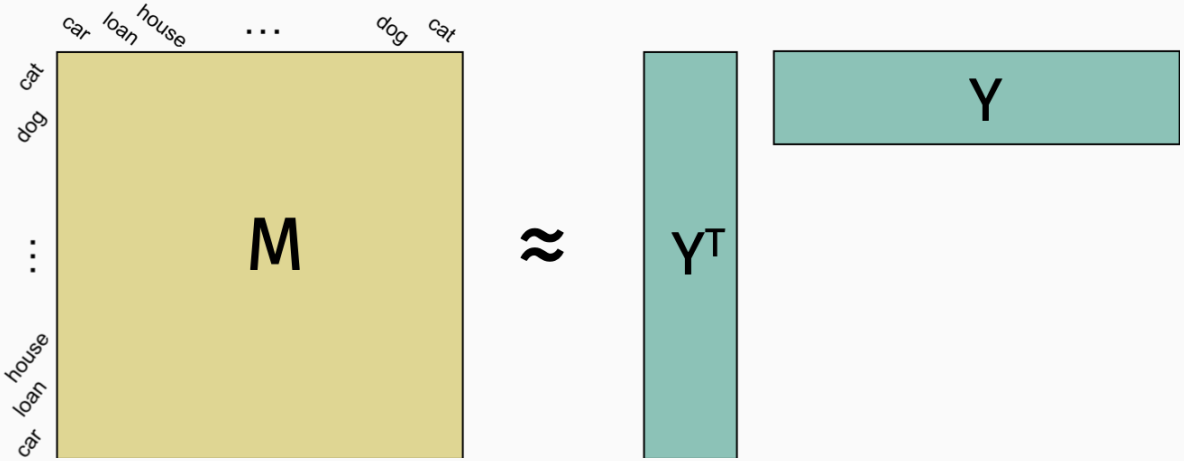
Word embeddings

General word embedding recipe:

1. Choose similarity metric $k(word_i, word_j)$ which can be computed for any pair of words.
2. Construct similarity matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ with $\mathbf{M}_{i,j} = k(word_i, word_j)$.
3. Find low rank approximation $\mathbf{M} \approx \mathbf{Y}^T\mathbf{Y}$ where $\mathbf{Y} \in \mathbb{R}^{k \times n}$.
4. Columns of $\mathbf{Y}$ are word embedding vectors.

We expect that $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ will be larger for more similar words.

Word embeddings



How do current state-of-the-art methods differ from LSA?

- Similarity based on co-occurrence in smaller chunks of words. E.g. in sentences or in any consecutive sequences of 3, 4, or 10 words.
- Usually transformed in non-linear way. E.g. $k(word_i, word_j) = \frac{p(i,j)}{p(i)p(j)}$ where $p(i,j)$ is the frequency both i, j appeared together, and $p(i)$, $p(j)$ is the frequency either one appeared.

Modern word embeddings

Computing word similarities for "window size" 4:

The girl walks to her dog to the park.
It can take a long time to park your car in NYC.
The dog park is always crowded on Saturdays.

The girl walks to her dog to the park.
It can take a long time to park your car in NYC.
The dog park is always crowded on Saturdays.

The girl walks to her dog to the park.
It can take a long time to park your car in NYC.
The dog park is always crowded on Saturdays.

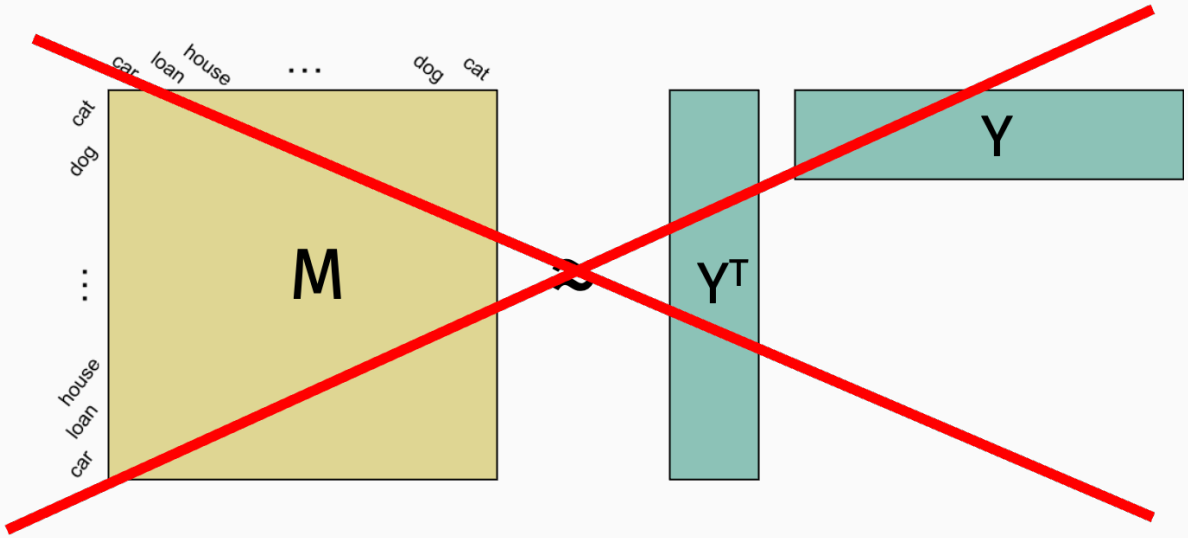
	dog	park	crowded	the
dog	0	2	0	3
park	2	0	1	2
crowded	0	1	0	0
the	3	2	0	0

Modern word embeddings

Current state of the art models: `GloVE`, `word2vec`.

- `word2vec` was originally presented as a shallow neural network model, but it is equivalent to matrix factorization method (Levy, Goldberg 2014).
- For `word2vec`, similarity metric is the "point-wise mutual information": $\log \frac{p(i,j)}{p(i)p(j)}$.

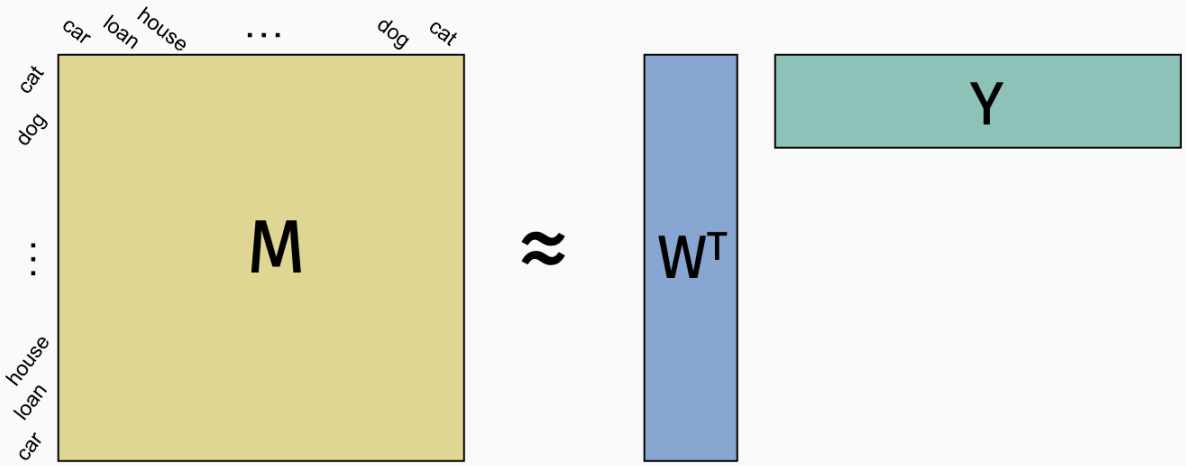
Caveat about factorization



SVD will not return a symmetric factorization in general. In fact, if $\mathbf{M}$ is not positive semidefinite[1] then the optimal low-rank approximation does not have this form.

[1] I.e., $k(word_i, word_j)$ is not a positive semidefinite kernel.

Caveat about factorization



- For each word i we get a left and right embedding vector $\mathbf{w}_i$ and $\mathbf{y}_i$. It's reasonable to just use one or the other.
- If $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ is large and positive, we expect that $\mathbf{y}_i$ and $\mathbf{y}_j$ have similar similarity scores with other words, so they typically are still related words.
- Another option is to use as your features for a word the concatenation $[\mathbf{w}_i, \mathbf{y}_i]$

Easiest way to use word embeddings

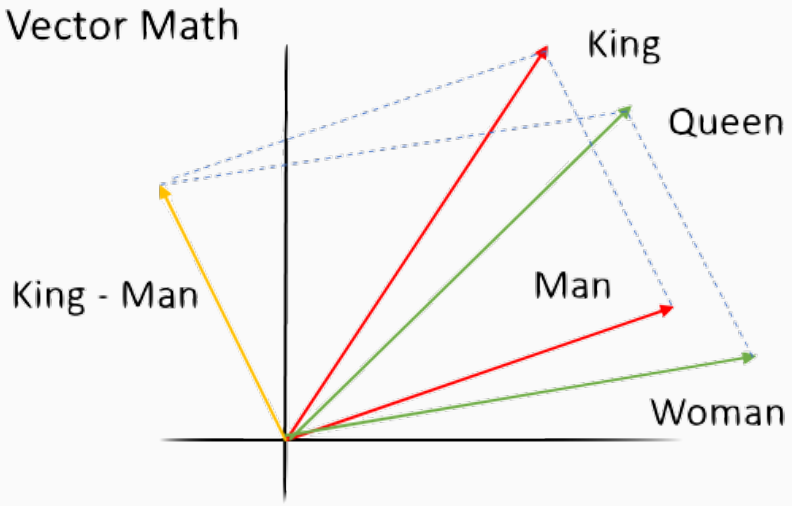
Lots of pre-trained word vectors are available online:

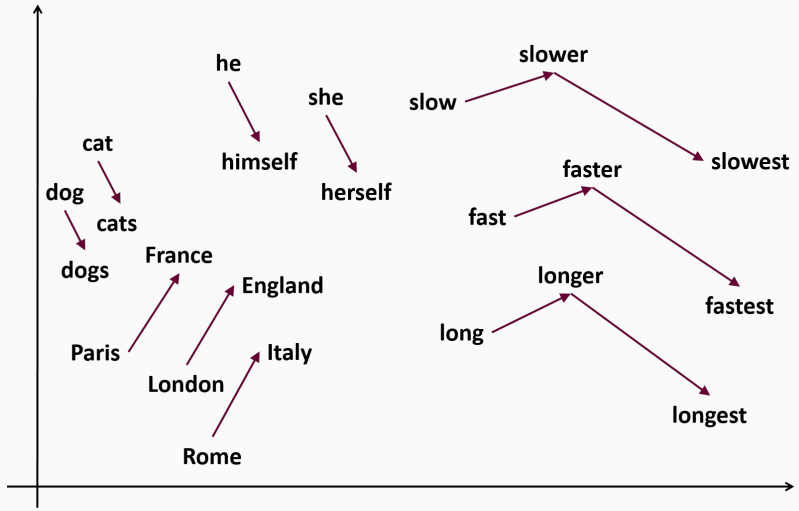
- **Original gloVe website:**
  `https://nlp.stanford.edu/projects/glove/`.
- **Compilation of many sources:**
  `https://github.com/3Top/word2vec-api`

Word embeddings math

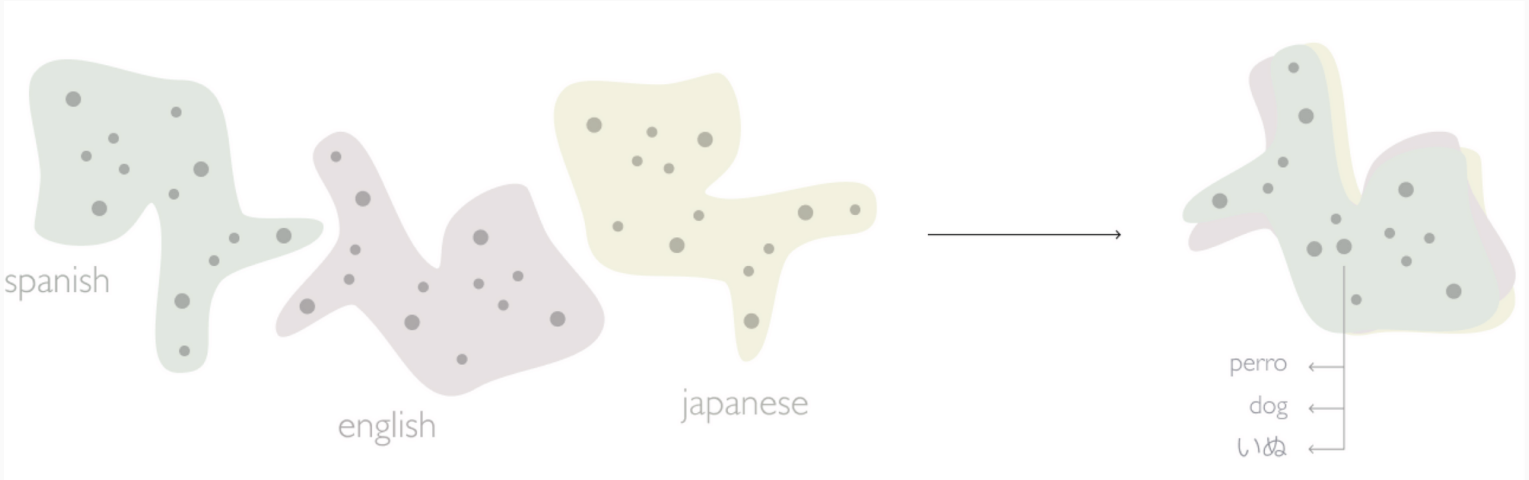
Lots of cool demos for what can be done with these embeddings.

E.g. "vector math" to solve analogies.





Forward looking application: unsupervised translation



- Train word-embeddings for languages separately. Obtain lowish dimensional point clouds of words.
- Perform rotation/alignment to match up these point clouds.
- Equivalent words should land on top of each other.

No needs for labeled training data like translated pairs of sentences!

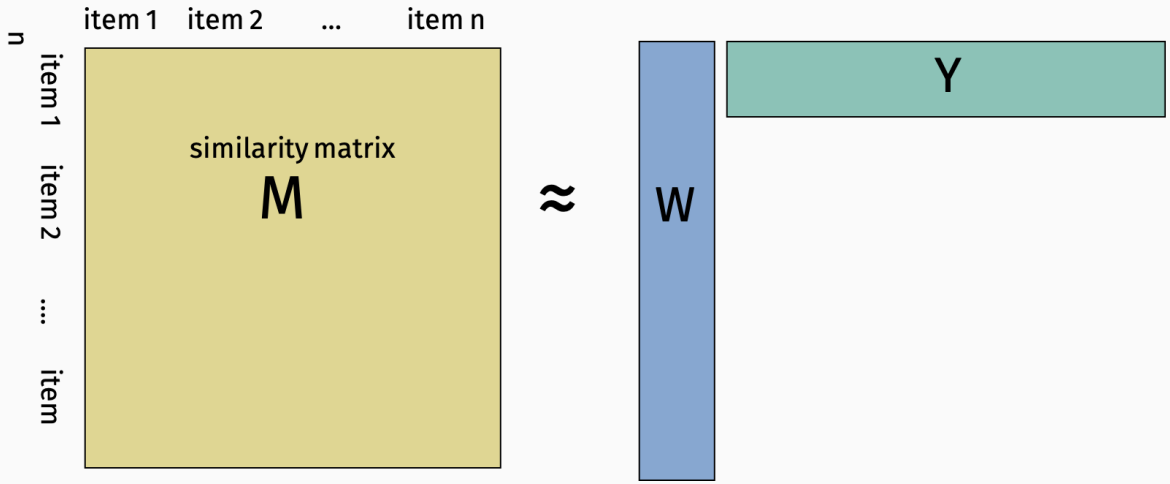
Forward looking application: unsupervised translation

Why not monkey or whale language?

Earth Species Project (www.earthspecies.org), CETI Project (www.projectceti.org)

Semantic embeddings

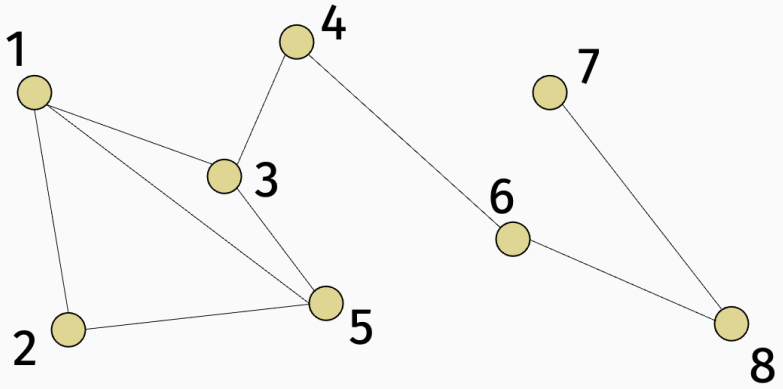
The same approach used for word embeddings can be used to obtain meaningful numerical features for any other data where there is a natural notion of similarity.



For example, the items could be nodes in a social network graph. Maybe be want to predict an individuals age, level of interest in a particular topic, political leaning, etc.

Node embeddings

Generate random walks (e.g. "sentences" of nodes) and measure similarity by node co-occurence frequency.

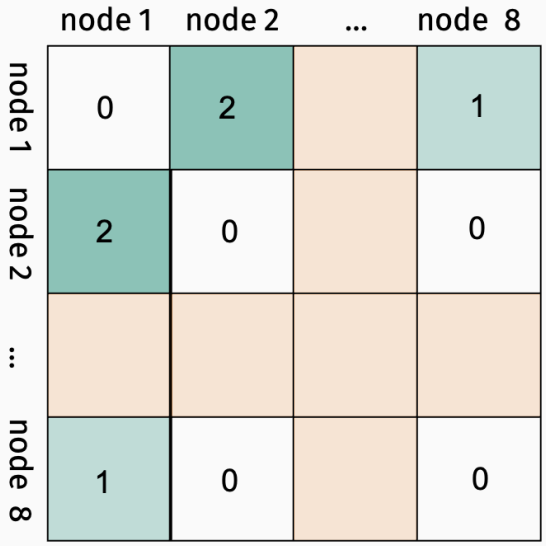


1, 3, 4, 4, 5, 2, 1, 2, 5
6, 8, 6, 4, 3, 1, 5, 3, 4
7, 8, 6, 8, 7, 8, 6, 8, 6
⋮
4, 6, 8, 6, 4, 3, 1, 2, 5

Node embeddings

Again typically normalized and apply a non-linearity (e.g. log) as in word embeddings.

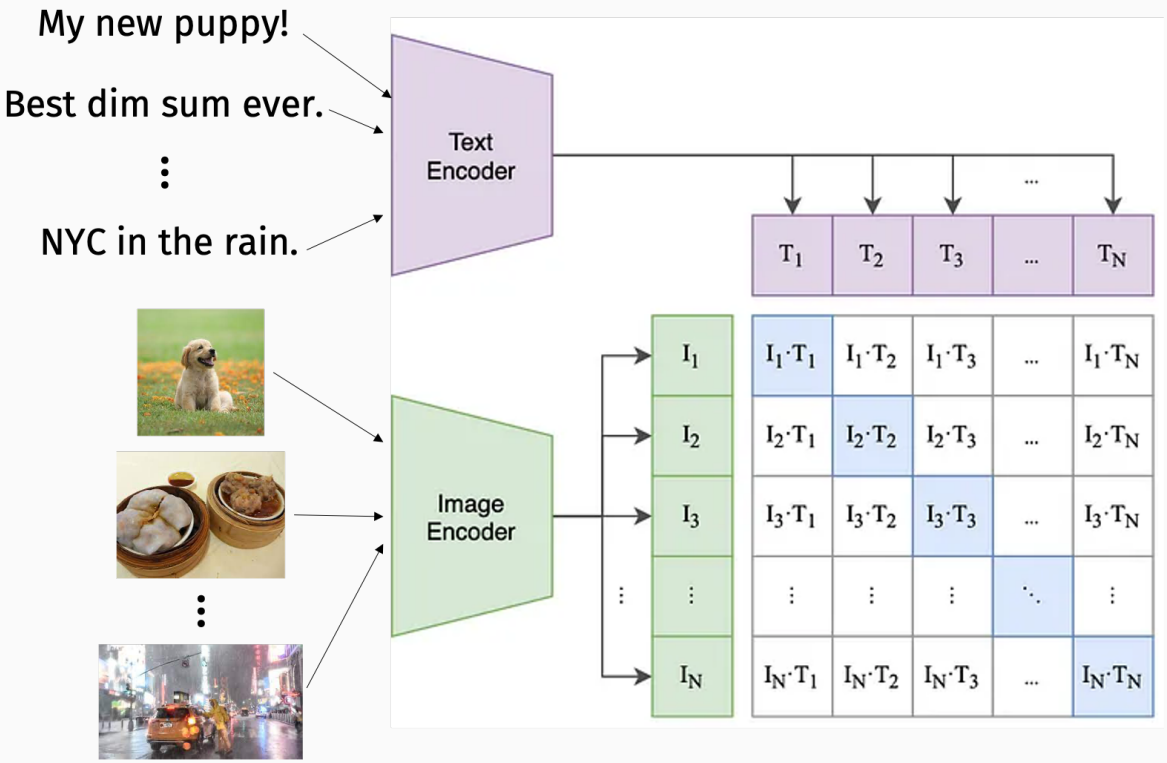
1, 3, 4, 4, 5, 2, 1, 2, 5
6, 8, 6, 4, 3, 1, 5, 3, 4
7, 8, 6, 8, 7, 8, 6, 8, 6
⋮
4, 6, 8, 6, 4, 3, 1, 2, 5



Popular implementations: `DeepWalk`, `Node2Vec`. Again initially derived as simple neural network models, but are equivalent to matrix-factorization (Qiu et al. 2018).

Bimodal embeddings

We can also create embeddings that represent different types of data. OpenAI's CLIP architecture:



Goal: Train embedding architectures so that $\langle \mathbf{T}_i, \mathbf{I}_j \rangle$ are similar if image and sentence are similar.

Contrastive Language–Image Pre-training (CLIP)

What do we use as ground truth similarities during training? Sample a batch of sentence/image pairs[2] and just use identity matrix.



My new puppy!	1	0	0
Best dim sum ever.	0	1	0
NYC in the rain.	0	0	1

This is called contrastive learning. Train unmatched text/image pairs to have nearly orthogonal embedding vectors.

[2] CLIP was trained on 400 million text-image pairs scraped from the internet.

CLIP for zero-shot learning

Learning Transferable Visual Models From Natural Language Supervision

Alec Radford [*1] Jong Wook Kim [*1] Chris Hallacy [1] Aditya Ramesh [1] Gabriel Goh [1] Sandhini Agarwal [1]
Girish Sastry [1] Amanda Askell [1] Pamela Mishkin [1] Jack Clark [1] Gretchen Krueger [1] Ilya Sutskever [1]

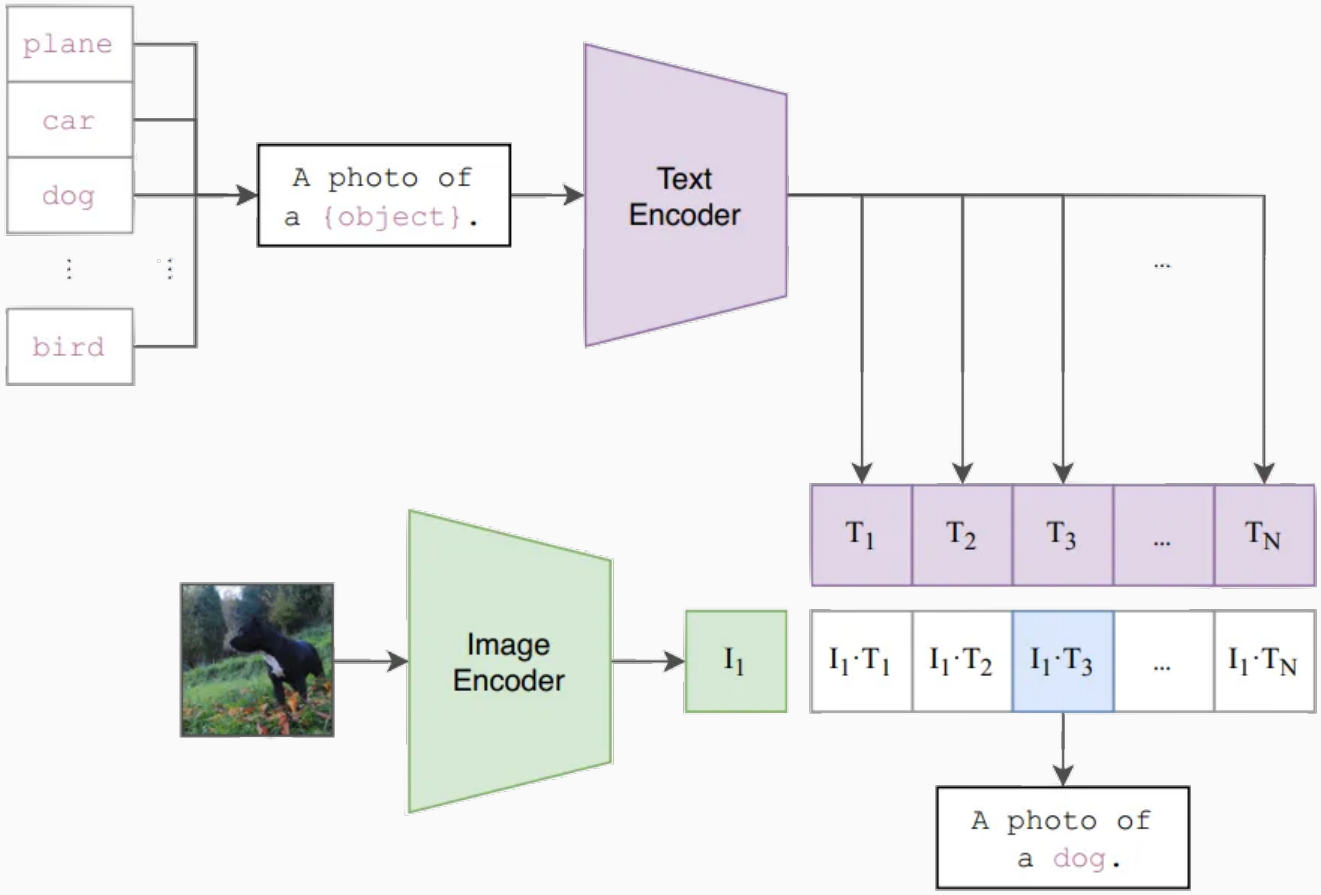
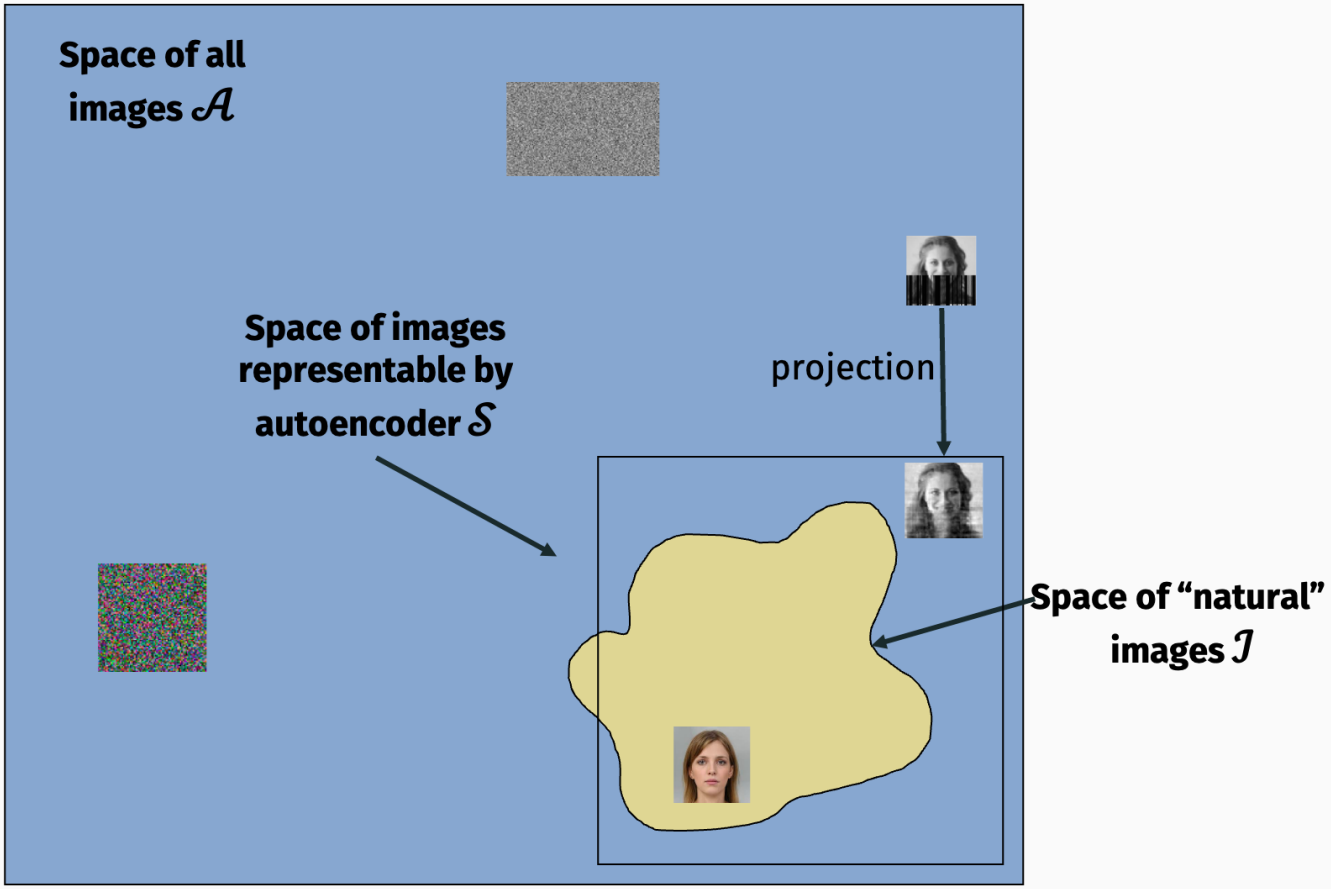


Image Synthesis

Autoencoders learn compressed representations

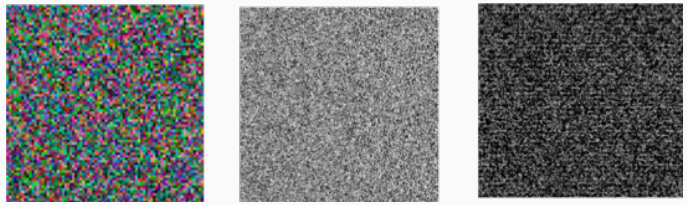


$f(\mathbf{x}) = d(e(\mathbf{x}))$ projects an image $\mathbf{x}$ closer to the space of natural images.

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 $\mathcal{A}$.

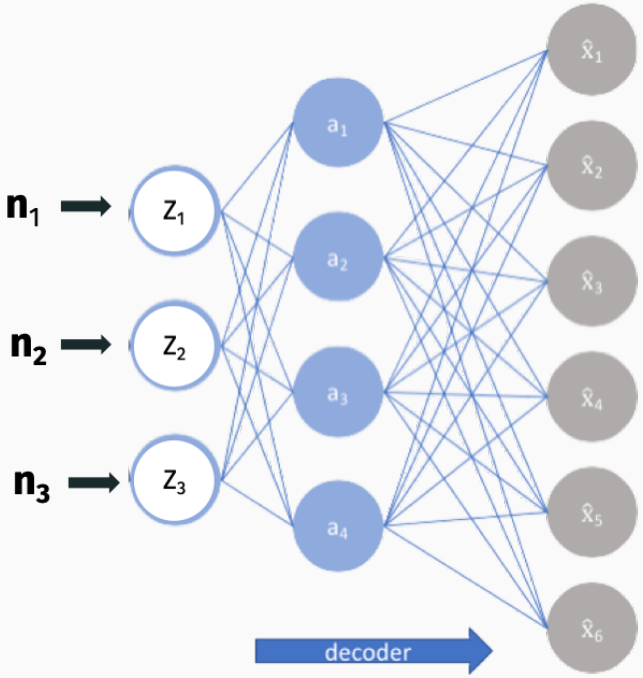


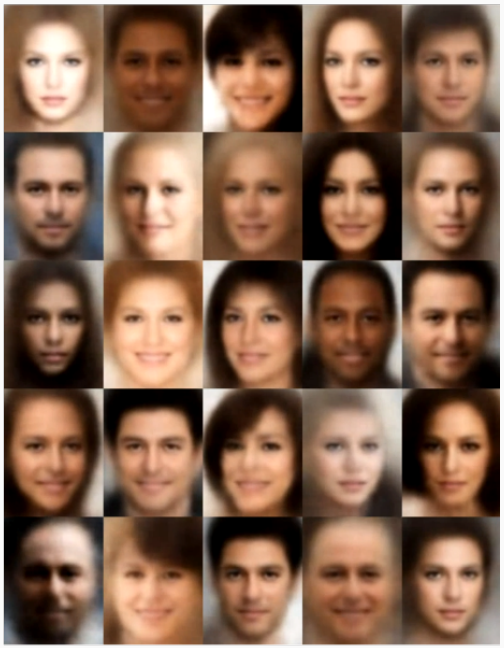
- **Option 2**: Draw $\mathbf{x}$ randomly from $\mathcal{S}$, the space of images representable by the autoencoder.

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

Autoencoders for data generation

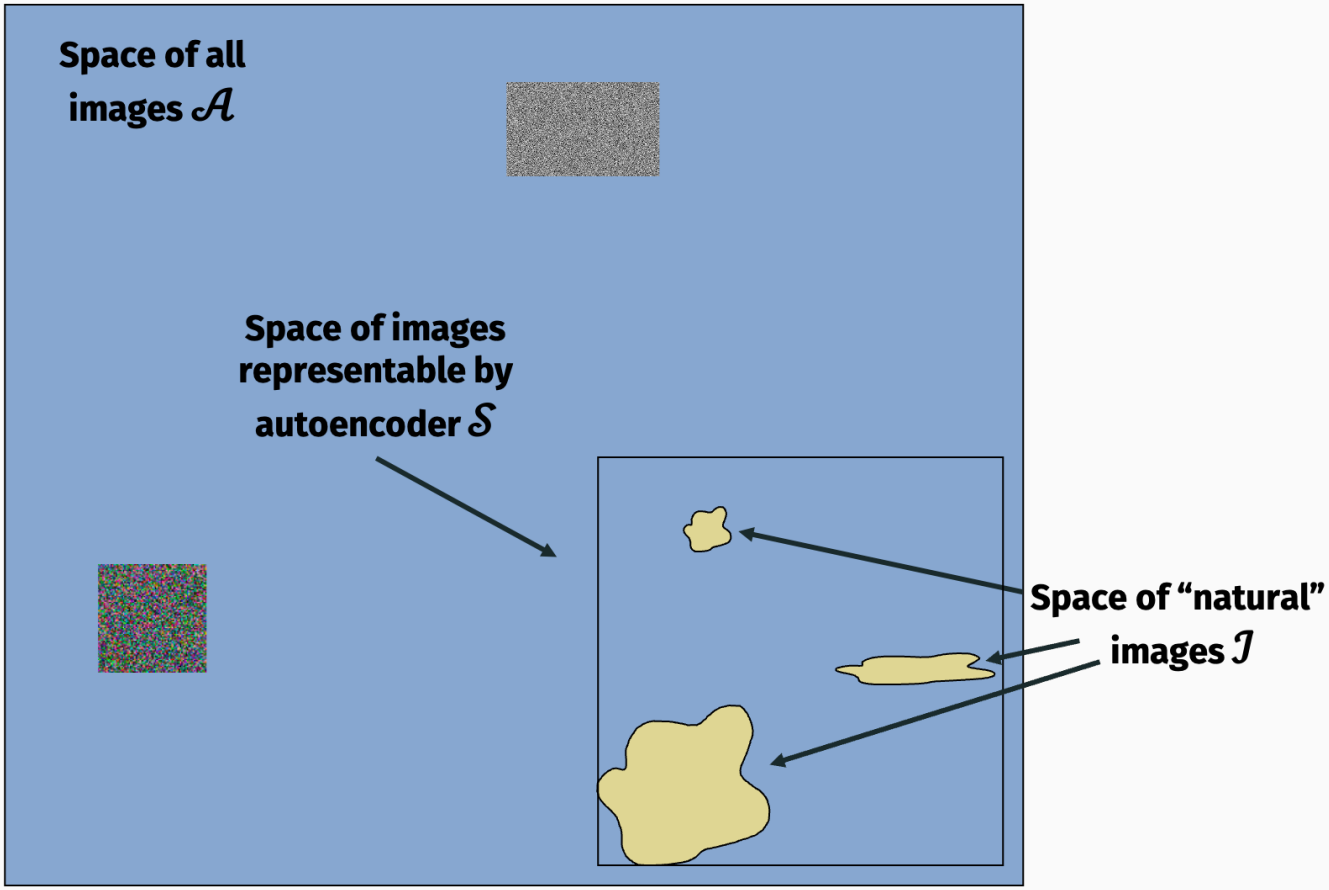
Autoencoder approach to generative ML: Feed random inputs into decode to produce random realistic outputs.





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

Autoencoders for data generation

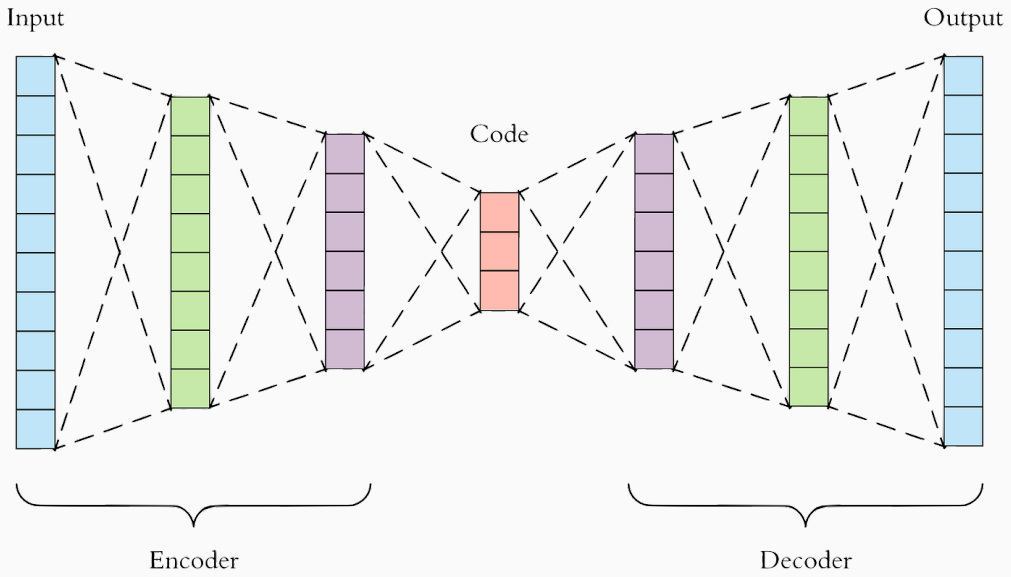


Variational Auto-Encoders (VAEs) attempt to resolve this issue.

Variational AutoEncoders (VAEs)

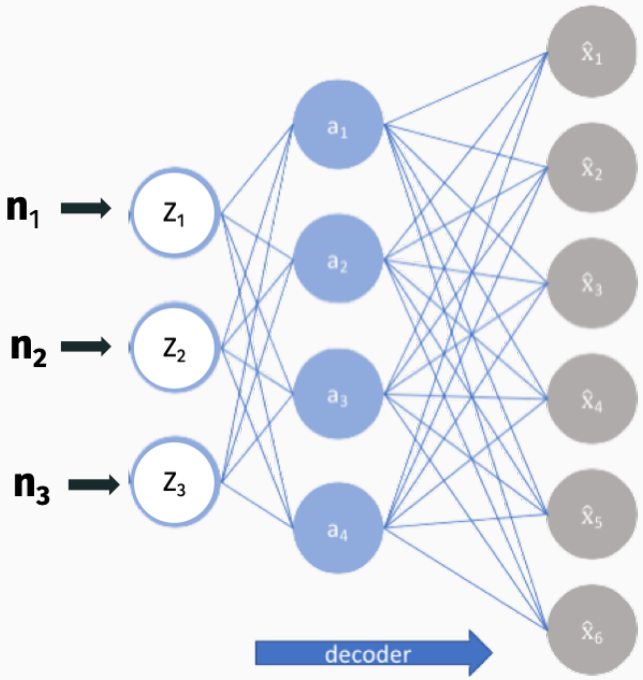
VAEs attempt to resolve this issue. Basic ideas:

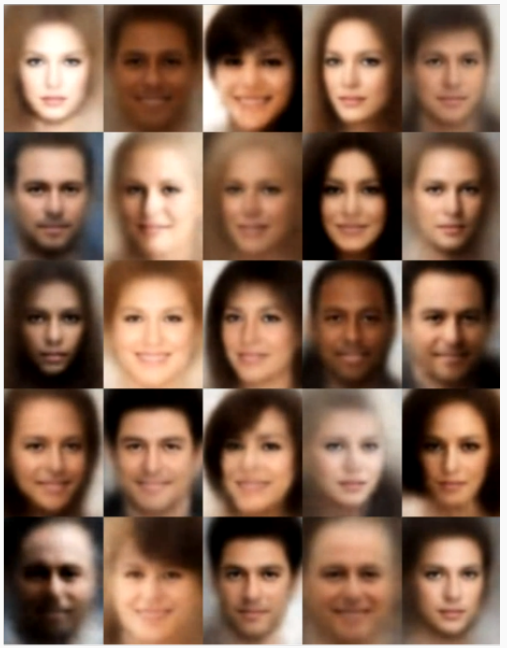
- Instead of mapping inputs to a single latent vector, VAEs map them to a probability distribution in the latent space (e.g., a Gaussian distribution)
- Add noise during training.
- Add penalty term so that distribution of code vectors generated looks like mean 0, variance 1 Gaussian.



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).





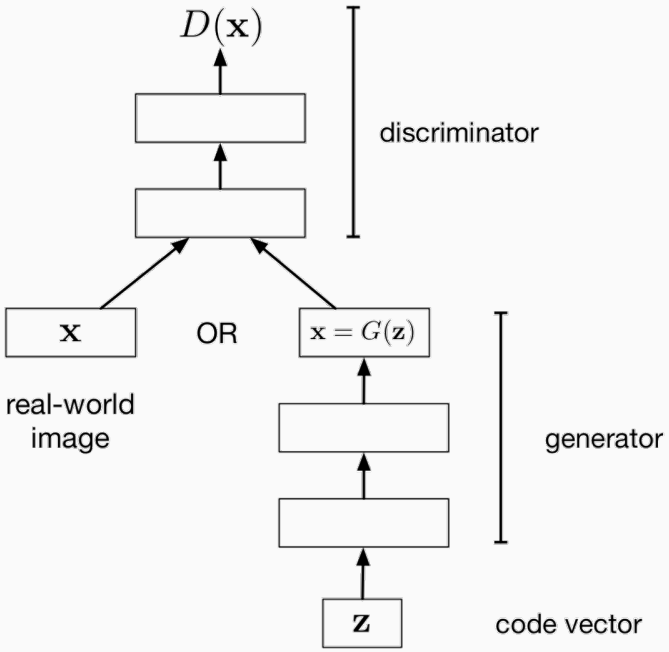
Generative Adversarial Networks (GANs)

Lots of efforts to hand-design regularizers that penalize images that don't look realisitic 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)

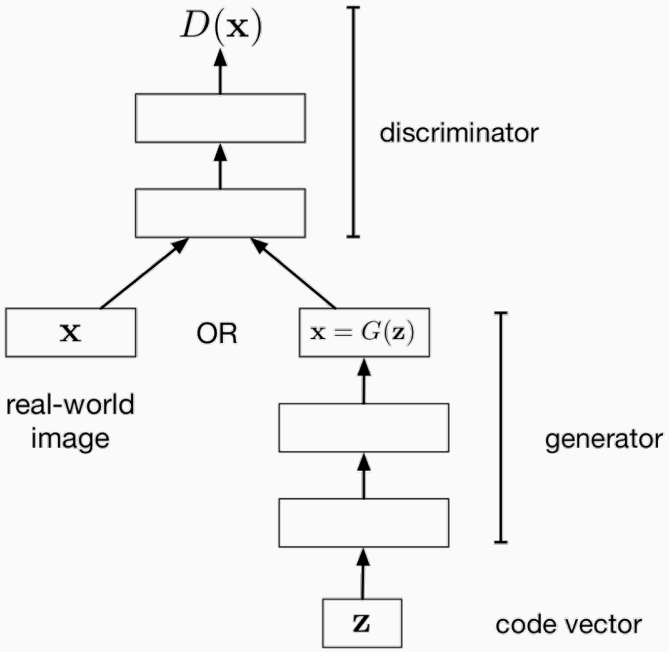


Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be real images and let $\mathbf{z}_1, \ldots, \mathbf{z}_m$ be random code vectors. The goal of the discriminator is to output a number between $[0, 1]$ which is close to 0 if the image is fake, close to 1 if it's real.

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

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

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 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.

Diffusion

Auto-encoder/GAN approach: Input noise, map directly to image.

Diffusion: Slowly move from noise to image.

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

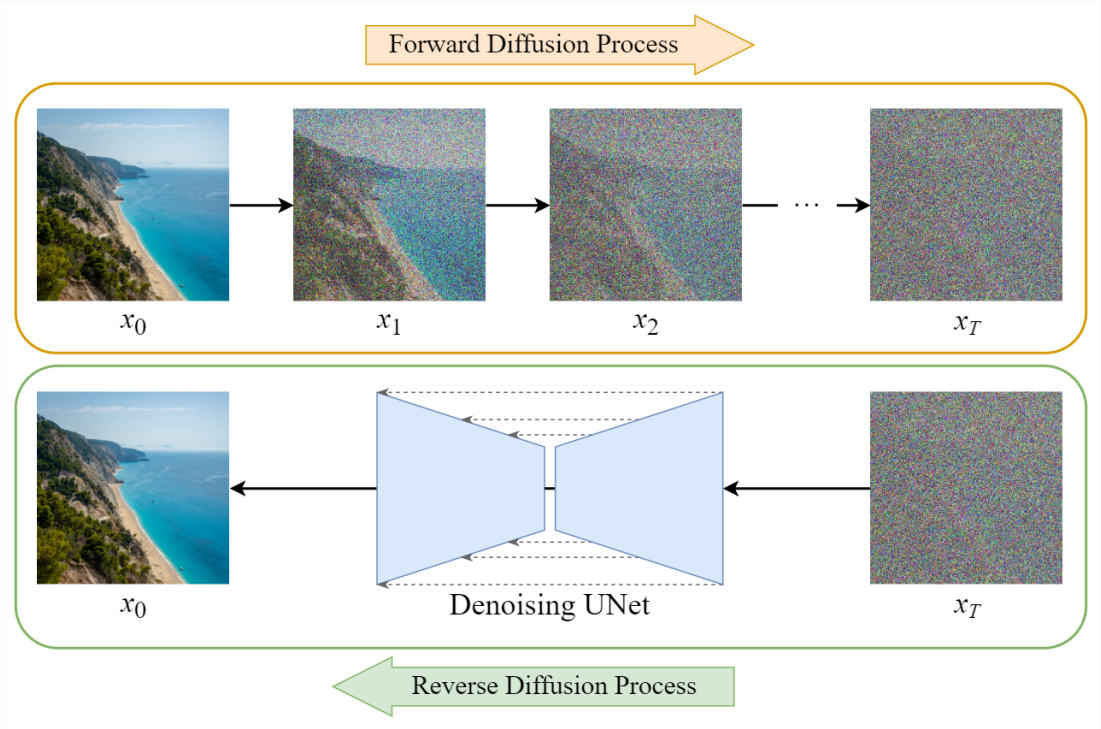
Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

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.

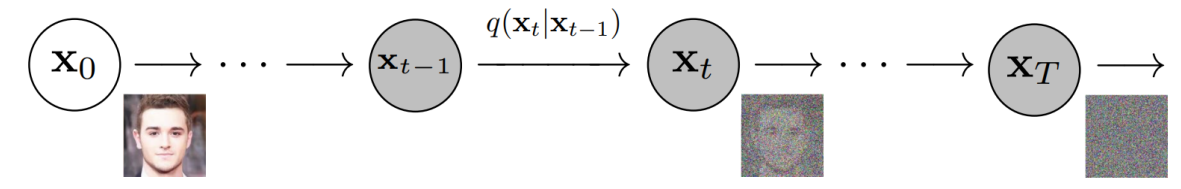


Key Question: How do we predict and reverse noise effectively?

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:

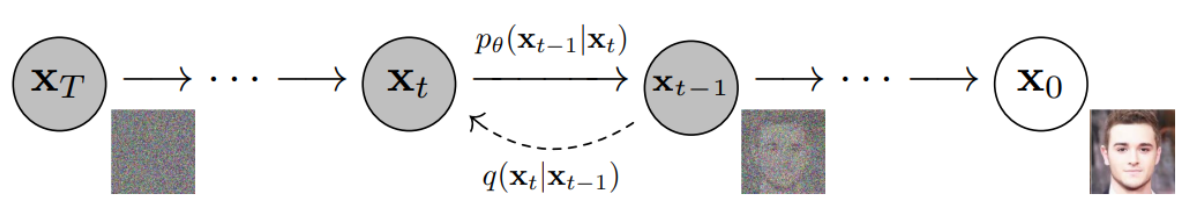
$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

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

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:

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

Training process

Data Preparation:

- Use large datasets of images $\mathbf{x}_0$.

Noise Addition:

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon$$

Model Training:

- Train $\epsilon_\theta(\mathbf{x}_t, t)$ to predict the noise.

Loss Function:

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

Image generation using diffusion

Sampling Process:

1. Start with pure noise $\mathbf{x}_T$.
2. Iteratively denoise using:

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sqrt{\Sigma_\theta} z, \quad z \sim \mathcal{N}(0, I)$$

3. Final output: a clean image $\mathbf{x}_0$.

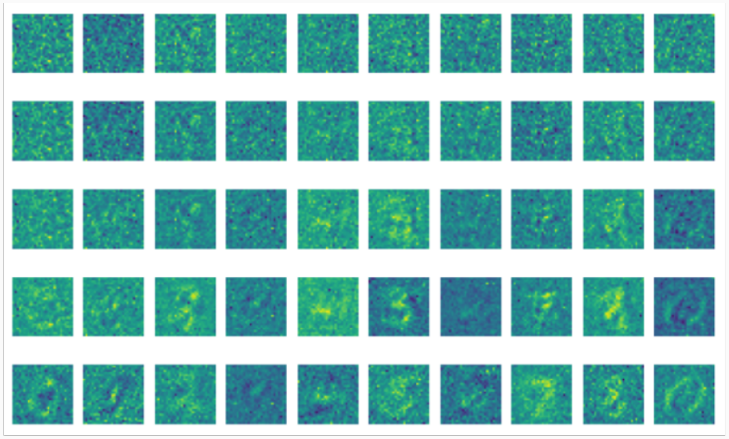
Semantic embeddings + diffusion

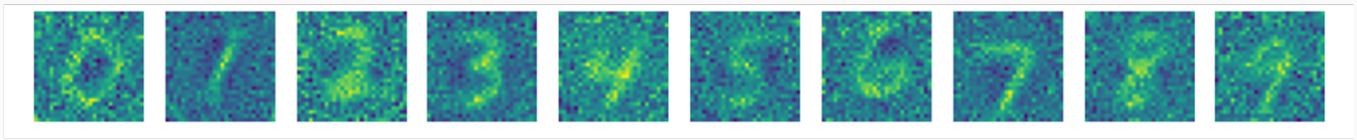
Text to image synthesis: Dall-E, Imagen, Stable Diffusion

"A chair that looks like a pineapple"

Diffusion

A demo for generating digits by training on MNIST.





Challenges

Some challenges:

- Slow inference (many denoising steps).
- Computationally expensive training. Can we store the entire dataset on a single server?
- Ethical and responsible AI practices.
  - Individual privacy
  - Bias
  - Fairness

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.

1 Introduction

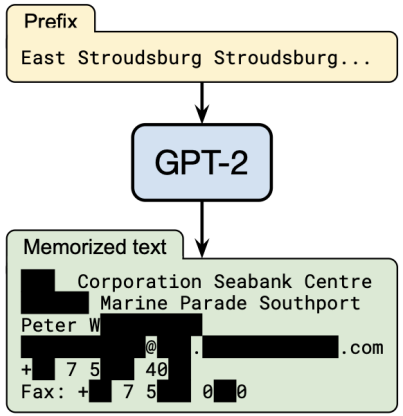


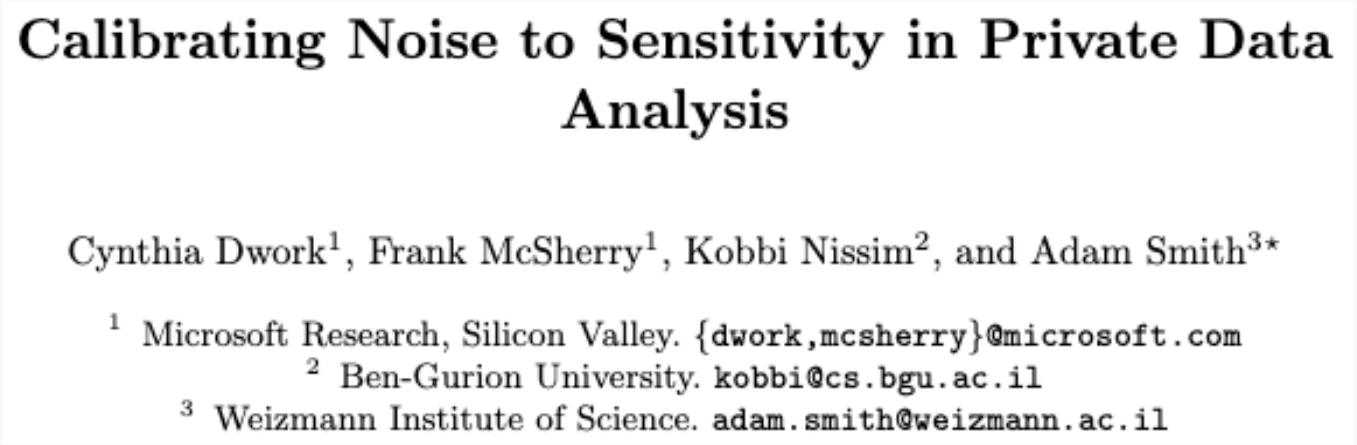
Figure 1: **Our extraction attack.** Given query access to a neural network language model, we extract an individual person's name, email address, phone number, fax number, and physical address. The example in this figure shows information that is all accurate so we redact it to protect privacy.

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:

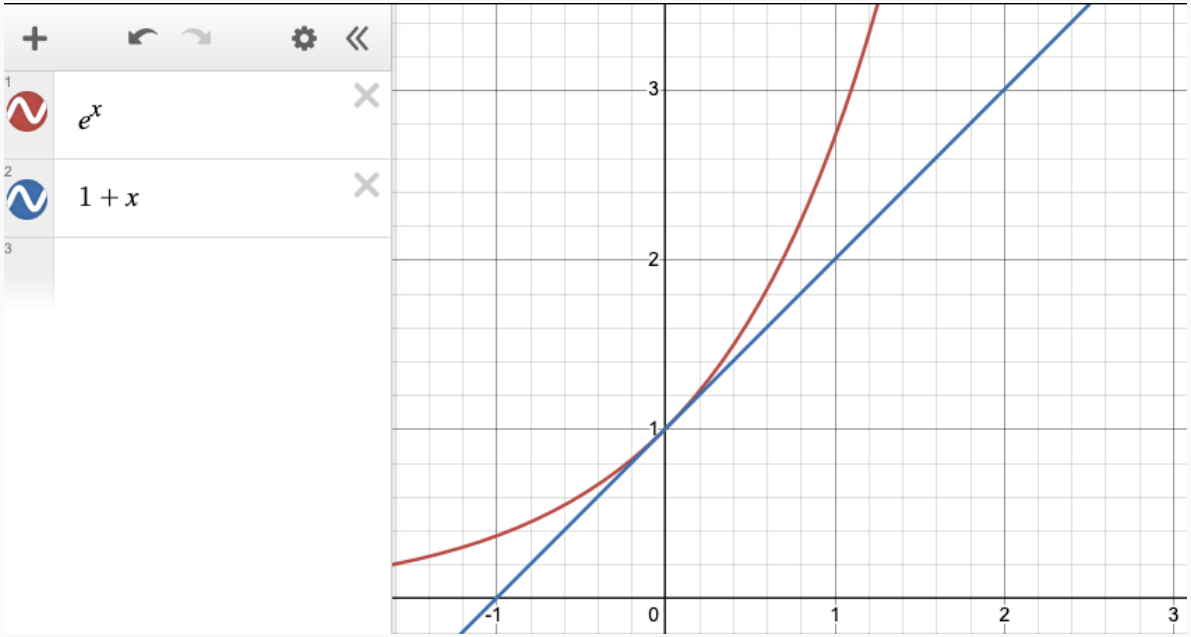
$$\frac{1}{n} \sum_{i=1}^{n} x_i.$$

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 privacy and accuracy.

What type of noise and how much?

Theorem (Laplace Mechanism)

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

Proof: For any possible output z,

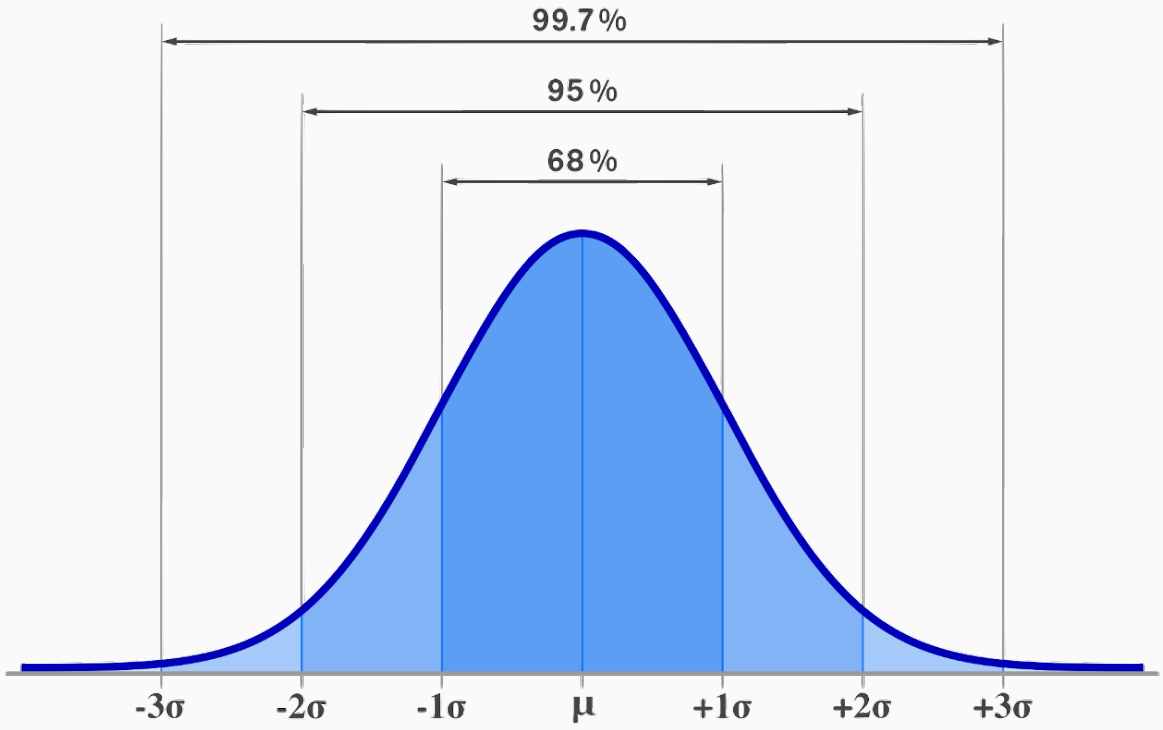
- $\Pr[\mathcal{A}(\mathbf{X}) = z] = \frac{1}{2(\Delta_Q/\epsilon)} e^{-|Q(\mathbf{X})-z|/(\Delta_Q/\epsilon)}$
- $\Pr[\mathcal{A}(\mathbf{X}') = z] = \frac{1}{2(\Delta_Q/\epsilon)} e^{-|Q(\mathbf{X}')-z|/(\Delta_Q/\epsilon)}$

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

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

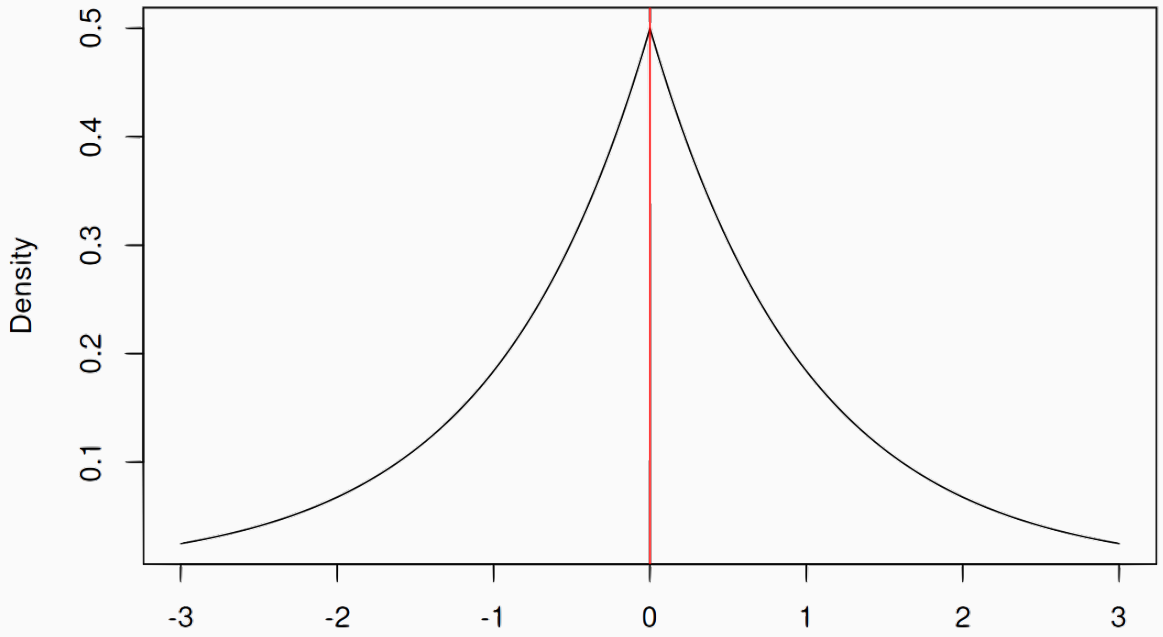
What do we pay in terms of accuracy?

$Lap(b)$ 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.

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

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.

Federated (Decentralized) Learning

Some challenges with centralized settings

- Privacy concern: in many applications individuals do not trust the central server. Individuals want to keep their raw data local
- Computational concern: collecting all the data at one central server and doing computation could be infeasible.

Federated Learning

Federated Learning: A decentralized learning paradigm where data remains local while models are trained collaboratively.

Communication-Efficient Learning of Deep Networks from Decentralized Data

H. Brendan McMahan **Eider Moore** **Daniel Ramage** **Seth Hampson** **Blaise Agüera y Arcas**
Google, Inc., 651 N 34th St., Seattle, WA 98103 USA

Federated Learning

- Typical loss function in machine learning:

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

  where $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ are the training data point.

- In the FL setting each client has its own local data points and its own local loss function.
- The loss can be broken down to sum of the clients' local losses.

$$L(\boldsymbol{\beta}) = \sum_{j \in \mathcal{P}_1} \ell_1(\boldsymbol{\beta}, \mathbf{x}_j, y_j) + \sum_{j \in \mathcal{P}_2} \ell_2(\boldsymbol{\beta}, \mathbf{x}_j, y_j) + ... + \sum_{j \in \mathcal{P}_K} \ell_K(\boldsymbol{\beta}, \mathbf{x}_j, y_j)$$

Federated Learning

We assume there are K clients over which the data is partitioned, with $\mathcal{P}_k$ the set of indexes of data points on client k, with $n_k = |\mathcal{P}_k|$.

Objective

$$\min_{\boldsymbol{\beta}} F(\boldsymbol{\beta}) = \sum_{k=1}^{K} \frac{n_k}{n} f_k(\boldsymbol{\beta})$$

where $f_k(\boldsymbol{\beta}) = \frac{1}{n_k} \sum_{j \in \mathcal{P}_k} L_k(\boldsymbol{\beta}, \mathbf{x}_j, y_j)$ is a user-specified loss function on client k local training dataset.

Federated SGD

Algorithmic framework:

- Communication rounds between the server and clients
- At each communication round the server broadcast the current global model to clients
- Clients, in parallel, do local SGD step
- Clients send their updates to the central server
- The server aggregates the updates and updates the global model.

FedSGD

Basic FedSGD algorithm:

Server chooses a starting model $\boldsymbol{\beta}^{(0)}$.

For $i = 0, \ldots, T$:

- Server broadcast the current model $\boldsymbol{\beta}^{(i)}$ to clients
  - Clients in parallel do:

    Compute $\nabla L_k(\boldsymbol{\beta}^{(i)}) = \frac{1}{n_k} \sum_{j \in \mathcal{P}_k} \nabla \ell_k(\boldsymbol{\beta}^{(i)}, \mathbf{x}_j, y_j)$

    Local GD update $\boldsymbol{\beta}_k^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla L_k(\boldsymbol{\beta}^{(i)})$

    Send $\boldsymbol{\beta}_k^{(i+1)}$ to server
- Server updates $\boldsymbol{\beta}^{(i+1)} = \sum_{k=1}^{K} \frac{n_k}{n} \boldsymbol{\beta}_k^{(i+1)}$

Return $\boldsymbol{\beta}^{(T)}$.

Advantages and challenges of FL

Advantages:

- Preserves data privacy by keeping data local.
- Enables collaborative training across multiple organizations or devices.
- Reduces risks of centralized data breaches.
- Facilitates training on diverse, real-world data without data sharing.

Challenges:

- Communication overhead between clients and server.
- Handling heterogeneous (non-iid) data distributions.
- Ensuring fairness across participants with varying data quality or quantity.
- Potentially high computational demands on client sides.