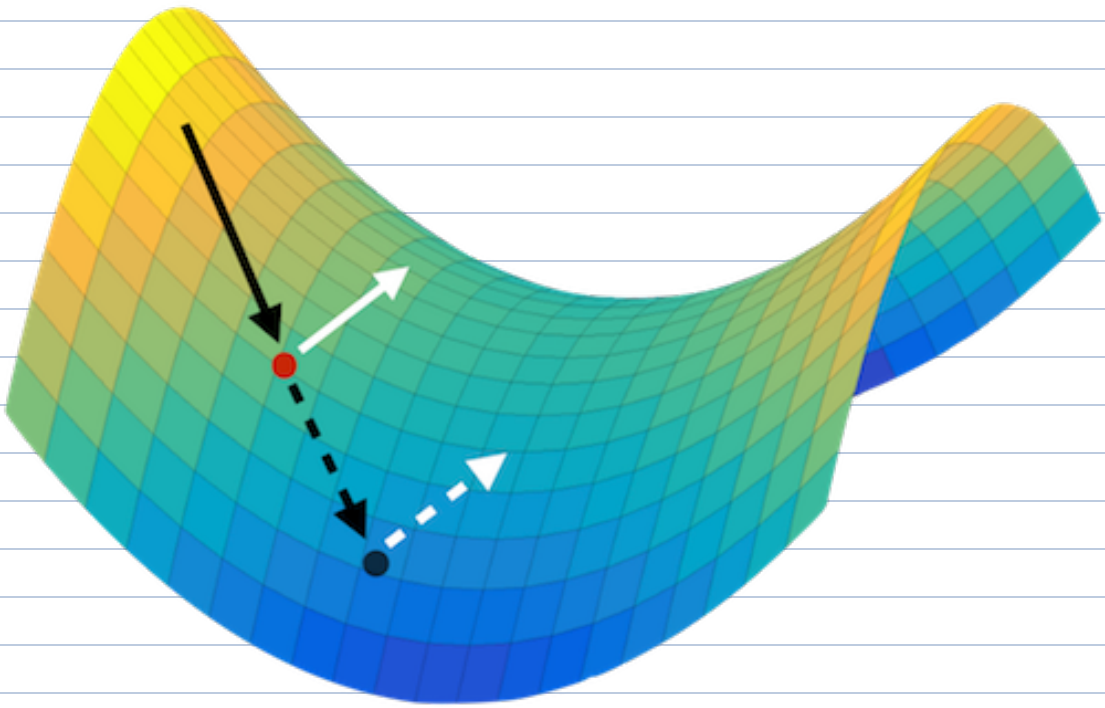
Algorithmic

Machine Learning



What is learning?

- Planning problems, classification, generative models
- Supervised learning: our data has labels
- Unsupervised learning: our data has no labels
- "Self-Supervised Learning": using un-labeled data to generate labeled data.
- Many others: (Semi-Supervised, Active Learning, Reinforcement Learning)

Modeling Supervised Learning:

Dataset: $(x_1, y_1) \dots (x_n, y_n)$, where x belongs to a domain d and y belongs to a label L

Goal: Given a new $x \in X$, we want to predict its label.

How do we ensure that their is a relationship between train and test. We assume there is an underlying ground truth distribution on training and test examples. We also assume that there is a class of functions that we are going to use to make predictions.

Learning as Optimization:

- We have an underlying D on X, labels L.
- Dataset: $(x_1, y_1), \cdots (x_n, y_n)$
- Output: "Parametrized hypothesis class: Θ" → Θ is the set of parameters that decide the model

Measuring Performance:

- Loss Function: $\ell: d \times d \to \mathbb{R}$, takes two labels and outputs a score depending on the two labels.

Empirical Risk Minimization (ERM):

Input: $(x_1, y_1), (x_2, y_2) \dots, (x_n, y_n)$

Output: $\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \ell(h_\theta(x_i), y_i)$

Θ: set of all allowable parameters

Training Loss

"Find the set of parameters from all of the allowable parameters that minimizes the loss function"

$h_\theta \equiv$ predicting model with parameters θ.

$1/n \equiv$ is used to take the average so we can compare across datasets.

In the above we test on examples we already saw, so how do we ensure we aren't memorizing the data?

Ideally: $\min_{\theta \in \Theta} \mathbb{E}_{x \sim D} [\ell(h_\theta(x), y]$ → Population Loss, what is the loss when we see an example we have never seen before, statistical machine learning gives relations between training and population loss.

Example #1 (ERM):

$X = \mathbb{R}^d$, $L = \mathbb{R}$, parametric family: Linear predictors ($\Theta \equiv \mathbb{R}^d$) which means: $h_\theta(x) = \sum_{i=1}^{d} \theta_i x_i = \theta_1 x_1 + \dots + \theta_n x_n$

Least Square Regression: $\ell: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$: $\ell(a,b) = (a-b)^2$: $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_\theta(x_i), y_i) = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{d} \theta_j x_j - y_i \right)^2$

$$\Rightarrow \langle \theta, x \rangle = \sum_{j=1}^{d} \theta_j x_j \Rightarrow \frac{1}{n} \sum_{i=1}^{n} (\langle \theta, x_i \rangle - y_i)^2$$

Suppose our loss function is: $\ell: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$: $\ell(a,b) = (a-b)^2$: $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} |\langle \theta, x_i \rangle - y_i|$

Example #2 (ERM):

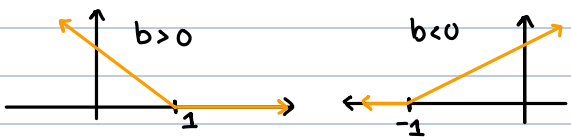
What happens when our labels are no-longer continuos but discrete instead?

$L = \{0, 1\}$ ⟶ parametric family: half-spaces

Half-spaces: $h_\theta(x) = \begin{cases} 1 & \text{if } \langle \theta, x \rangle > t \\ 0 & \text{if } \langle \theta, x \rangle \leq t \end{cases}$ t is a threshold value

def Hinge Loss:

$$\ell(a,b) = \begin{cases} \max(0, 1-a) & \text{if } b > 0 \\ \max(0, 1+a) & \text{if } b < 0 \end{cases}$$

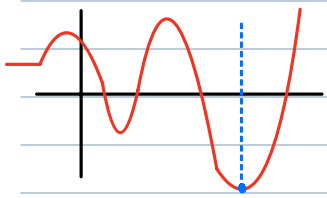


It is useful in the case that the label is discrete but our prediction is not.

We will say learning is equivilent to solving the optimization problem.

Let us make a generalization we will say there exists a function $L: \mathbb{R}^d \to \mathbb{R}$ let us do unconstrained optimization to find θ to minimize $L(\theta)$.

Let us consider a loss function with respect to one parameter:



We want to find the lowest point on the curve, but we cannot see ahead or backwards.

Greedy Approach: move in the direction of steepest descent

The direction of steepest decent is to move in the negative gradient: $-\nabla L(\theta)$

Gradient = $\nabla L(\theta) = \left(\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_1}, \dots, \frac{\partial L}{\partial \theta_d}\right)$

Gradient Descent Algorithm (GD):

Take a starting point θ_0

For $i = 1, \dots, T$:

$\theta_i = \theta_{i-1} - \eta \nabla L(\theta_{i-1})$

↳ step size

We can apply this to any ERM problem as long as we can calculate the gradient of the loss function.

In ERM: $\nabla L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(h_\theta(x_i), y_i)$

Gradient Descent is nice as long as the function is differentiable, there is another form we could apply if L is non-differentiable we use the subgradient:

Sub-gradient: A direction g is a sub gradient of f at x_0 if $\forall x$ $f(x_0) + \langle g, x - x_0 \rangle \leq f(x)$, this only needs to hold locally.

line with slope g: $f(x_0) + g(x - x_0)$

Questions about gradient descent:

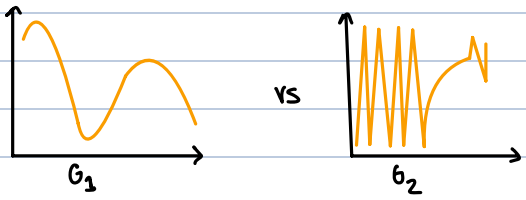
1) Will it find a true minimum
2) How many steps does GD take
3) How to compute the gradient
4) Choosing the step size
5) Choosing the starting point
6) When does it stop?
7) If the gradient is difficult to compute, are there any surrogates?

Analyzing GD:

We have a function $f:\mathbb{R}^d \to \mathbb{R}$ we want $\min_x f(x)$

GD: For $i=1,2,\dots,T$:

$$x_{i+1} = x_i - \eta \nabla f(x_i)$$



With gradient descent there is a higher likely hood that in G_2 if we dont have the right step size it would keep changing between two peaks. Hence we want a less "spikey" function for GD.

Lipschitzness: f is L-Lipschitz ($f:\mathbb{R}^d \to \mathbb{R}$) if $\forall x,y \quad |f(x)-f(y)| \le L \cdot \|x-y\|_2$ ↷ euclidian distance

eg. $f(x)=10x$ $|f(x)-f(y)| = 10|x-y|$ hence f is 10 lipschitz

eg $f(x)=|x|$; f is now 1-Lipschitz

Lipschitzness talks about how much values differ by in a function

Smoothness: f is β-smooth if $\forall x,y \quad \|\nabla f(x) - \nabla f(y)\|_2 \le \beta \cdot \|x-y\|_2$ ("Lipschitz for the gradient")

(Recall: $u \in \mathbb{R}^d$, $\|u\|_2 = (\sum_i u_i^2)^{1/2}$)

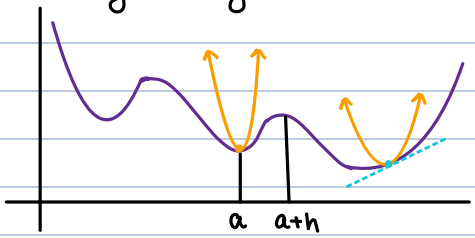
eg: $f(x)=|x|$ is 1-lipschitz, but f is not smooth: $\nabla f(-0.001) = -1$ and $\nabla f(0.001) = 1$

eg: $f:\mathbb{R}\to\mathbb{R}$. $f(x) = ax^2+bx$ and $f'(x) = 2ax+b$; $f(x)-f(y) = ax^2+bx-ay^2-by = a(x-y)(x+y)+b(x-y)$

we observe that f is not lipschitz for any finite f. But we absorve that $|f'(x)-f'(y)| = 2a|x-y|$. Then f is $(2a)$ smooth.

Lemma: $f:\mathbb{R}^d \to \mathbb{R}$ and is β-smooth if $\eta \le \frac{1}{\beta}$, then $f(x_{i+1}) \le f(x_i) - \frac{\eta}{2}\|\nabla f(x_i)\|_2^2 \le f(x_i)$

"Monotonicity of GD" given this we will always makee progress with this step size.



Smoothness upperbound: f is β smooth then

$\forall a,h \quad f(a+h) \le f(a) + f'(a)h + \frac{\beta}{2}h^2$

The proof is based on Taylors Theorm...

let $x_{i+1} = x_i - \eta \nabla f(x_i)$. Then we have $f(x_{i+1}) = f(x_i - \eta f'(x_i)) \le f(x_i) + h \cdot f'(x_i) + \frac{\beta}{2} \cdot h^2$

$= f(x_i) - (\eta \cdot f'(x_i)) f'(x_i) + \frac{\beta}{2}(-\eta f'(x_i))^2 = f(x_i) - \eta\left(1 - \frac{\eta\beta}{2}\right) \cdot f'(x_i)^2$ hence if $\beta \le \frac{1}{\eta}$ then we get the above lemma. (This proves lemma when $d=1$.

A higher dimensional lemma:

Smoothness upper-bound: let f be β-smooth, $f:\mathbb{R}^d \to \mathbb{R}$. $\forall a,h \quad f(a+h) \le f(a) + \langle \nabla f(a), h\rangle + \frac{\beta}{2}\cdot\|h\|_2^2$

Product becomes inner product & squares become square of the normal direction.

Recall: ⚠ $\langle u,v\rangle = \sum_i u_i v_i$

The gradient descent update provides us with $x_{i+1} = x_i - \eta \cdot \nabla f(x_i)$. Hence we have that $f(x_{i+1}) = f(x_i - \eta \nabla f(x_i))$. By the smoothness upperbound we get that:

$$\begin{aligned} f(x_{i+1}) &\le f(x_i) + \langle \nabla f(x_i), -\eta \nabla f(x_i)\rangle + \beta/2 \, \|-\eta \nabla f(x_i)\|^2 \\ &= f(x_i) - \eta \langle \nabla f(x_i), \nabla f(x_i)\rangle + \beta\eta^2/2 \, \|\nabla f(x_i)\|^2 \\ &= f(x_i) - \eta \|\nabla f(x_i)\|^2 + \frac{\eta^2\beta}{2}\|\nabla f(x_i)\|_2^2 \\ &= f(x_i) - \eta\left(1 - \frac{\eta\beta}{2}\right)\|\nabla f(x_i)\|^2 \end{aligned}$$

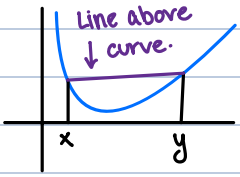
GD makes progress as long as $\eta \leq \frac{1}{\beta}$, one option is to tailor the step size based on the local smoothness of the curve. Other practical tricks:

1). Find largest η such that $f(x_i - \eta \nabla f(x_i)) \leq f(x_i) - \frac{\eta}{2}\|\nabla f(x_i)\|_2^2$

2). Backtracking line Search, to figure out the best η to use. (This is one way to find #1 (eg try 100 then 50... so on and so forth).

Convex Functions:

has many applications in optimization. A function $f: \mathbb{R}^d \to \mathbb{R}$ is convex if in 2d the tangent lined or in 3d the tangent plane at any point is below the curve. More formally $\forall x,y : f(\frac{x+y}{2}) \leq \frac{1}{2}f(x) + \frac{1}{2}f(y)$. Needs to hold for all points.



Equivilent to saying: $\forall x,y \ \ \lambda \in (0,1) \ \ f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$

& Equivilent to saying: $\forall x,y \ \ f(x) + \langle \nabla f(x), y - x \rangle \leq f(y)$

→ equation of tangent line/plane

Properties of convex functions:

① f, g are convex then $f+g$ is convex

② f is convex $\Rightarrow a \cdot f$ is convex if $a > 0$.

③ $g: \mathbb{R} \to \mathbb{R}$, and a vector w in $\mathbb{R}^d$. then $g_w: \mathbb{R}^d \to \mathbb{R}$, $g_w(x) = g(\langle w, x \rangle)$ then if g is convex then g_w is convex.

Eg: e^x, $\ln(x)$, x^2, $\|x\|^2$ over $\mathbb{R}^d$, $|x|$ are all convex

Generalizing this too $f(\frac{x_1 + x_2 + \ldots + x_n}{n}) \leq \frac{f(x_1) + \ldots + f(x_n)}{n}$

Revisiting Least Squares Regression:

- Dataset: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Hypothesis class: $h_\theta(x) = \langle \theta, x \rangle$
- loss function: $\ell(a,b) = (a-b)^2$

We observe ℓ is a convex function: $g(z) = (z - y_i)^2$ is convex hence: $(\langle \theta, x_i \rangle - y_i)^2$ is also convex as a function of θ.

Hence the ERM: $\frac{1}{n}\sum_{i=1}^{n}(\langle \theta, x_i \rangle - y_i)^2$ is also convex, so it is now a convex optimization problem.

Even if we were too plug in a loss function $|\langle \theta, x_i \rangle - y_i|$ would still be convex.

LASSO:

We can put a "penalty" on weights being too large, : $\lambda(|\theta_1| + |\theta_2| + \ldots + |\theta_d|)$ or in the case that their are multiple optimal θ's this method helps make our solution less sparse meaning we have fewer non-zero enteries.

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\langle \theta, x_i \rangle - y_i)^2 + \lambda\|\theta\|_1 \quad \text{and this is also still convex.}$$

"Convex optimization": optimizing a convex function.

Optimality of GD in convex functions:

Fixed constant *

Theorem: $f: \mathbb{R}^d \to \mathbb{R}$. f is β-smooth and convex, then $\eta \leq \frac{1}{\beta}$ we get $f(x_k) \leq f(x_*) + \frac{\beta \|x_0 - x_*\|^2}{2k}$. where x_* is the global minimum. $(\|x_0 - x_*\|)^2$ quantifies our starting point with respect to the global optimum.

x_* is the optimal solution, x_0 is starting point of gradient descent

Example: To get within ϵ of the optimum we need $k = \frac{\beta \cdot \|x_0 - x_*\|^2}{2 \cdot \epsilon}$ then $f(x_k) \leq f(x_*) + \epsilon$. To get $1/\epsilon$ close we need to run $1/\epsilon$ times.

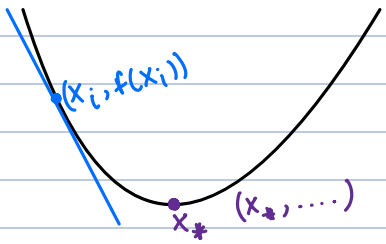
We want to show this to be optimal:

By smoothness we know:

$$f(x_{i+1}) \leq f(x_i) - \frac{\eta}{2} \|\nabla f(x_i)\|_2^2$$

By convexity we know:

$$f(x_i) + \langle \nabla f(x_i), x_* - x_i \rangle \leq f(x_*)$$



We already know that GD makes progress from monoticity we need to quantify k.

The relationship between x_i and x_* is: $f(x_i) + \langle \nabla f(x_i), x_* - x_i \rangle \leq f(x_*)$

$\Rightarrow$ By addition: $f(x_{i+1}) + f(x_i) + \langle \nabla f(x_i), x_* - x_i \rangle \leq f(x_i) - \frac{\eta}{2}\|\nabla f(x_i)\|^2 + f(x_*)$

$\Rightarrow f(x_{i+1}) + \langle \nabla f(x_i), x_* - x_i \rangle \leq f(x_*) - \frac{\eta}{2}\|\nabla f(x_i)\|^2$

$\Rightarrow f(x_{i+1}) - f(x_*) \leq \frac{-\eta}{2}\|\nabla f(x_i)\|^2 - \langle \nabla f(x_i), x_i - x_* \rangle$

$\Rightarrow$ From GD we know: $x_{i+1} = x_i - \eta \nabla f(x_i) \Rightarrow \nabla f(x_i) = \frac{x_i - x_{i+1}}{\eta}$

$\Rightarrow \leq \frac{-\eta}{2}\|\frac{x_i - x_{i+1}}{\eta}\|^2 - \frac{1}{\eta}\langle x_i - x_{i+1}, x_i - x_* \rangle$

$\Rightarrow \leq \frac{1}{\eta}\langle x_{i+1} - x_i, x_i - x_* \rangle - \frac{\eta}{2}\|\frac{x_{i+1} - x_i}{\eta}\|^2$

We now get that: $f(x_{i+1}) - f(x_*) \leq \frac{1}{2\eta}\left[2\langle x_{i+1} - x_i, x_i - x_* \rangle - \|x_{i+1} - x_i\|^2\right]$

From linear algebra we know that: $2\langle u, v \rangle = \|u\|^2 + \|v\|^2 - \|u - v\|^2$

$\Rightarrow f(x_{i+1}) - f(x_*) \leq \frac{1}{2\eta}\left[\|x_{i+1} - x_i\|^2 + \|x_i - x_*\|^2 - \|x_{i+1} - x_*\|^2 - \|x_{i+1} - x_i\|^2\right]$

$\Rightarrow \leq \frac{1}{2\eta}\left[\|x_i - x_*\|^2 - \|x_{i+1} - x_*\|^2\right]$ → this says that the change in function value to how close we are to the optimum depends the distance between the iterate to the optimum.

Let's write some iterations:

$$f(x_1) - f(x_*) \leq \frac{1}{2\eta}\left[\|x_0 - x_*\|^2 - \|x_1 - x_*\|^2\right]$$

$$f(x_2) - f(x_*) \leq \frac{1}{2\eta}\left[\|x_1 - x_*\|^2 - \|x_2 - x_*\|^2\right]$$

$$\vdots$$

$$f(x_{k+1}) - f(x_*) \leq \frac{1}{2\eta}\left[\|x_k - x_*\|^2 - \|x_{k+1} - x_*\|^2\right]$$

This is a telescoping sum so if we add our values from 1 to K we get:

$$f(x_1) - f(x_*)) + (f(x_2 - f(x_*)) + \dots + (f(x_{k+1}) - f(x_*)) \leq \frac{1}{2\eta}\left[\|x_0 - x_*\|^2 - \|x_{k+1} - x_*\|^2\right] \leq \frac{1}{2\eta}\|x_0 - x_*\|^2$$

Since GD is monotone we observe that $(k+1)(f(x_{k+1} - f(x)) \leq (f(x) - f(x_*)) + (f(x_2) - f(x_*)) + \dots + (f(k_{+1}) - f(x_*))$

$\Rightarrow f(x_{k+1}) - f(x_*) \leq \frac{1}{2\eta}\left[\frac{\|x_0 - x_*\|^2}{k}\right]$ ← We sum k+1 things, and $f(x_{k+1}) - f(x_*)$ is the smallest

Thm 2: If f is Lipschitz and convex, we can get convergence of the form: $f(x_k) \leq f(x_*) + \frac{L \cdot \|x_0 - x_*\|}{\sqrt{k}}$

Requirements for GD:

→ All we need is the ability to compute gradients (first order methods only compute the first derivative)

First Order Methods of Optimization:

→ Suppose we have a sub-routine that computes $\nabla f(x)$ at any point x.

Nesterov's Accelerated Gradient Descent (NAGD)

we have gradient descent: $f(x_k) \leq f(x_*) + \frac{\|x_0 - x_*\|^2}{\beta \cdot k^2}$, so now to get within ε of the optimum

↳ NAGD takes $\frac{1}{\sqrt{\varepsilon}}$ iterations.

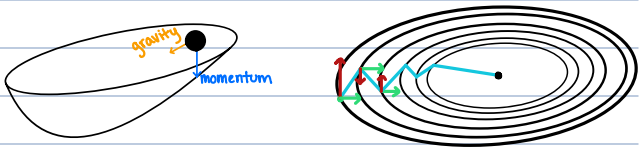
↳ This is also the best possible

Adding Momentum:

→ The idea is that our new position = old position + velocity. And we have the idea of increased inertia so we are less susceptible.

→ In regards to GD it is equivilent to keeping track of the previous magnitudes and having those influence our future steps.

→ We want to average out gradient directions that don't move in the right direction.



Observe that the gradient vectors:

↳ + ↓ + ↳ ⇒ → (In this case the vertical components cancel)

Formalizing NAGD:

The updates, for only momentum:

- $x_{i+1} = x_i - \eta \cdot g_i$
- $g_i = \alpha \cdot \nabla f(x_i) + (1-\alpha) g_{i-1}$, observe that $\alpha = 1$ it GD with no momentum, and $\alpha = 0$ is just momentum alone

  Next Step $\equiv \alpha$(Gradient) + $(1-\alpha)$(Old step)

Nestrov's Method is like momentum plus looking one step ahead.

The updates for NAGD:

- $x_{i+1} = y_i - \eta \nabla f(y_i)$
- $z_{i+1} = z_i - \eta_i \nabla f(y_i)$
- $y_{i+1} = \alpha \cdot z_i + (1-\alpha) x_i$

→

Theorem: f is convex and smooth then $f(x_k) \leq f(x_i) + \frac{2\beta \|x_0 - x_*\|^2}{k^2}$

we say a good setting for $\alpha_i = \frac{2}{i+3}$ and $\eta_i = \frac{(i+1)\eta}{2}$, where $\eta \leq 1/\beta$

The proof for NAGD will be on the next page...

Adaptive Learning Rates:

"Oscillations are bad". Idea: take bigger steps along directions where gradient is more stable.

The update would be that: $(x_{i+1})_j = (x_i)_j - \eta_j (g_i)_j$ we have a differrent update step size for each coordinate and direction.

One proposition was: $(\eta_i)_j \propto \frac{1}{|g_i|_j}$; called adagrad

↑ proportional

ADAGRAD:

- Start at x_0 and $(G_0)_i = 0$.
- For $i = 1, \ldots,$
  - $\rightarrow (G_i)_j = (G_{i-1})_j + (\nabla f(x_{i-1}))_j^2$
  - $\rightarrow (x_i)_j = (x_{i-1})_j - \eta_j (\nabla f(x_{i-1}))_j$ and $\eta_j = \frac{1}{\varepsilon + \sqrt{(G_i)_j}}$

Adagrad looks at changing the gradient in a particular direction of the gradient. A varient of this is RMS-Prop which uses a weighted average (so the first step will have less influence on the 100^{th} compared to the 10^{th}).
ADAM = RMS-PROP + "Momentum"

Least Squares Regression:

$\rightarrow L(w) = \frac{1}{n} \sum_{i=1}^{n} (\langle w, x_i \rangle - y_i)^2$

$\rightarrow \nabla L(w) = \frac{1}{n} \sum_{i=1}^{n} 2(\langle w, x_i \rangle - y_i) \cdot \nabla(\langle w, x_i \rangle - y_i) = \frac{1}{n} \sum_{i=1}^{n} 2(\langle w, x_i \rangle - y_i) \cdot x_i$

In terms of Matricies and how it would be coded up: $L(w) = \frac{1}{2} \|X \cdot w - y_i\|_2^2$ and $\nabla L(w) = \frac{2}{n} X^T (Xw - y)$

Analyzing the Cost of GD and NAGD?

$\frac{2}{n}(X^T)(Xw - y)$. It takes $O(nd)$ to compute $Xw - y$, and then the overall time complexity to compute $(X^T)(Xw - y)$ is $O(nd)$. Instead if we did $(X^T X)w - X^T y$ which has a time complexity of $O(d^2 \cdot n)$
Hence the cost per iteration in LSR is $O(nd)$ per iteration. Sometimes we look to minimize the cost per iteration rather than the number of iterations.

There exists a closed form solution for least squared: $w^* = (X^T X)^{-1} \cdot X^T y$ and using the formula takes $O(nd^2 + d^3)$ time.

Applying GD to general ERM:

$L(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_w(x_i), y_i)$ we have that $\boxed{\nabla L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{\partial \ell(h_w(x_i), y_i)}{\partial a} \right) \left(\nabla_w h_w(x_i) \right)}$

Since $\ell(a,b)$ is a bivariate function we first must calculate $\frac{\partial \ell(a,b)}{\partial a}$ and then substitute our parametrized linear family. Now the cost is at least $n \cdot d$
For example if we have 1 million images: $n = 10^6$ and $d = 250 \times 250$ so we have that $n \cdot d = 10^8 \cdot 25 \cdot 25$, such a high cost per iteration is an issue.

Stochastic Gradient Descent:

We make a trade-off: faster computation per iteration, and instead each iteration has an estimator for the gradient which is easier to compute but not as accurate.
We have a function $f: \mathbb{R}^d \rightarrow \mathbb{R}$. for every w, we have access to a random vector g_w such that $\mathbb{E}[g_w] = \nabla f(w)$.
where $\mathbb{E}$ is our estimator with a variance.
For a vector valued random variable z, $\text{var}(z) = \mathbb{E}[\|z - \mathbb{E}[z]\|^2]$

$\rightarrow$ Start with some x_0
$\rightarrow$ for $i = 1, \ldots,$:

$x_i = x_{i-1} - \eta g_{x_{i-1}}$ where $g_{x_{i-1}}$ is an estimator for $\nabla f(x_{i-1})$

ERM and SGD:

Let us say our estimator would be $g_w = \frac{\partial \ell}{\partial a}(h_w(x_i), y_i) \cdot \nabla_w h_w(x_i)$ where i is picked uniformely at random for $\{1,2,\dots,n\}$. We are picking a single random point.

Then $\mathbb{E}[g_w] = \frac{1}{n} \cdot \frac{\partial \ell}{\partial a}(h_w(x_1), y_1) \cdot \nabla_w h_w(x_1) + \frac{1}{n} \cdot \frac{\partial \ell}{\partial a}(h_w(x_2), y_2) \nabla_w h_w(x_2) + \dots + \frac{1}{n} \frac{\partial \ell}{\partial a}(h_w(x_n) y_n) \cdot \nabla_w h_w(x_n)$

$= \nabla_w L(w)$ the cost of computing the estimator is $O(d)$.

Mini-batch SGD:

→ Pick k random points $i_1, i_2, \dots, i_k \in \{1,2,\dots,n\}$ and $g_w = \frac{1}{k} \sum_{p=1}^{k} \frac{\partial \ell}{\partial a}(h_w(x_p), y_p) \nabla h_w(x_p)$.

the variance of mini-batch would be around $1/k \cdot$ var(only one sample).

In practice we permute our data-set once and then go over one example after the other. ⇒ Called an "Epoch"

Analyzing SGD:

We first remark that SGD is a random algorithm and monotonicity is not guaranteed.

Thm: f is β-smooth, convex. We have an estimator for gradients with variance $\le \sigma^2$.

$$\mathbb{E}[f(\bar{x})] \le f(x_*) + \frac{\|x_0 - x_k\|^2}{2\eta k} + \eta \cdot \sigma^2, \text{ where } \bar{x} = \frac{(x_0 + x_1 + \dots + x_k)}{k}.$$

Since SGD is not mono-tone our guarentee holds only on the average value for $\bar{x}$

Proving the Theorm:

- By smoothness upperbound: $f(x_{i+1}) \le f(x_i) + \langle \nabla f(x_i), x_{i+1} - x_i \rangle + \frac{\beta}{2} \|x_{i+1} - x_i\|_2^2$
- We have $x_{i+1} = x_i - \eta g$ where $(\mathbb{E}(g) = \nabla f(x_{i-1}))$

→ $f(x_{i+1}) \le f(x_i) + \langle \nabla f(x_i), -\eta g \rangle + \frac{\beta}{2} \|\eta \cdot g\|_2^2$.

Taking expectations: $\mathbb{E}[f(x_{i+1})] \le f(x_i) - \eta \langle \nabla f(x_i), \mathbb{E}[g] \rangle + \frac{\beta}{2} \eta^2 \cdot \mathbb{E}[\|g\|_2^2]$

→ $f(x_i) - \eta \langle \nabla f(x_i), \nabla f(x_i) \rangle + \frac{\beta \eta^2}{2} \mathbb{E}[\|g\|_2^2]$. If we have a real-valued r.v. $\mathrm{Var}(\delta) = \mathbb{E}[\delta^2] - (\mathbb{E}[\delta])^2$ and if this is applied per coordinate it is the same for vectors: $\mathrm{Var}(z) = \mathbb{E}[\|z\|^2] - \|\mathbb{E}[z]\|^2$

→ $f(x_i) - \eta \|\nabla f(x_i)\|^2 + \frac{\beta \eta^2}{2} (\|\nabla f(x_i)\|^2 + \mathrm{var}(g))$ and given $\eta \le 1/\beta$

$\le f(x_i) - \frac{\eta}{2} \|\nabla f(x_i)\|^2 + \frac{\eta}{2} \sigma^2$

Hence for SGD we have that:

$$\mathbb{E}[f(x_{i+1})] \le f(x_i) - \frac{\eta}{2} \|\nabla f(x_i)\|^2 + \frac{\eta}{2} \sigma^2$$

By convexity we knew that: $f(x_i) + \langle \nabla f(x_i), x_* - x_i \rangle \le f(x_*)$

Applying expectation we get that $\mathbb{E}[f(x_i)] + \mathbb{E}[\langle \nabla f(x_i), x_* - x_i \rangle] \le f(x_*)$

If we repeat the same process as GD we get that

$$\mathbb{E}[f(x_i)] - f(x_*) \le \mathbb{E}[\|x_{i-1} - x_*\|^2] - \mathbb{E}[\|x_i - x_*\|^2] + \frac{\eta \sigma^2}{2}$$

Then apply a telescoping sum to get:

$$\sum_{i=1}^{k} (\mathbb{E}[f(x_i)] - f(x_*)) \le \frac{1}{2\eta} \|x_0 - x_*\|^2 + \left(\frac{\eta \sigma^2}{2}\right) \cdot k$$

and we get that $\mathbb{E}[f(\frac{x_1 + \dots + x_k}{k})] \le f(x_*) + \frac{1}{2\eta} \frac{\|x_0 - x_*\|^2}{k} + \frac{\eta \sigma^2}{2}$

Choosing η for SGD:

We observe that we need a balanced η value since one is in the numerator while the other is in the denominator.

Thm: f is β smooth, and convex, and we choose $\eta = \sqrt{\frac{\|x_0 - x_*\|^2}{2k\sigma^2}}$, then we get a new lower convergence bound

$$\mathbb{E}(f(\bar{x})) = f(x_*) + \frac{2\|x_0 - x_*\|\sigma}{\sqrt{2k}}$$

Remark: If we have only estimators for the gradient, then $1/\sqrt{k}$ is the best possible convergence rate we would have.

Constrained Optimization!!

So far we have been working on unconstrained optimization is where we have $\min_{x \in \mathbb{R}^d}$. With constrained optimization we look to only have a particular set C: $\min_{x \in C} f(x)$. Genearl algorithms such as GD may not easily be applicabble.

Options:

1). "Barrier Methods" → we add a penalty term for crossing C

2). Projection Method: pick the closest point from X to the C: $\text{Proj}_C(x) = \arg\min_{y \in C} \|x - y\|_2$

Projected Gradient Descent (PGD):

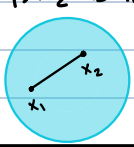
→ x_0

→ For $i = 1, \ldots,$:

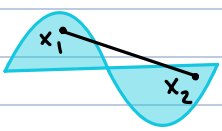
$$x_i = \text{Proj}_C(x_{i-1} - \eta \nabla f(x_{i-1}))$$

Convex Sets:

A set C is convex if $\forall x_1, x_2 \in C$, $\frac{x_1 + x_2}{2} \in C$. Which is the same as $\forall x_1, x_2 \in C$, the line segment connecting x_1, x_2 is in C.



Convex



Non-convex

All guarantees for GD, NAGD, SGD that apply for unconstrained optimization work with projections if C is convex.

Future Topics and Expansions:

→ Different norms, so far we only used euclidean distance for steepest descent. For example when working with graphs there exist better norms which allow for better gradient descent algorithms

→ Changing the "landscape". We transform the landscape to make it more nicer to optimize and then transform it back

→ Mirror Descent ... (changing variables and then doing GD)

→ Practical Tricks: (gradient clipping, adaptive learning rates, ...) that will help make gradient descent more practically applicable.

Online Learning:

→ We don't get to see all the data at once, some examples include recommendation systems, video-games learning the opponents strategy and then adapting, stock market predictions.

Day 1: x_1, $\hat{y}_1$, y_1, $\ell(\hat{y}_1, y_1)$

x_1: example; $\hat{y}_1$: prediction; y_1: truth; $\ell(\hat{y}_1, y_1)$: score

On day 2 we will have a different model so we need to have a way to compare models. We do not look at previous days data. What is a reasonable goal for online learning?

Mistake Bounded Model:

→ We work under the assumption that there is some truth function: $y = f_*(x)$. There is some unkown function that gives the right prediction.

→ We want an algorithm that makes a bounded number of mistakes (there is some structure to the truth)

Applying Online Learning to Halfspaces:

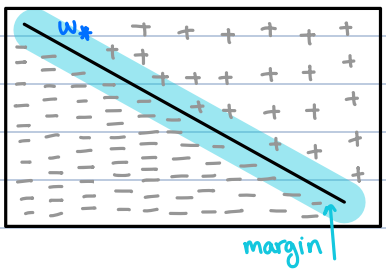
→ $f_*(x) = \text{sign}(\langle w_*, x \rangle)$ for some w_* our hypothesis class $h(x) = \{1 \text{ if } \langle w_*, x \rangle \geq 0 \text{ or } -1 \text{ else}\}$

→ having a margin will make it easier to learn:

Marging a Halfspace: $\gamma = \min_{x \in D} \frac{|\langle w_*, x \rangle|}{\|w_*\|_2 \|x\|_2}$, takes the smallest point away and compares it to the norm of x

→ Assume that $\|w_*\|_2 = 1$ and we normalize all examples X to have $\|x\|_2 = 1$

→ The width of the blue strip is gamma.



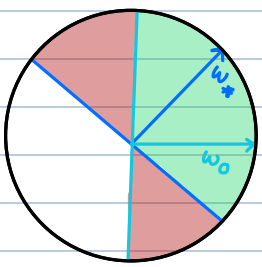
Perceptron:

We start with a random vector w_0

→ If our example is in the green, then we get the right prediction, if we are in the red we get the wrong prediction.

→ On Day i:

$$w_i = \begin{cases} w_{i-1} & \text{if no mistake } (\hat{y}_i, y_i) \\ w_{i-1} + x_i & \text{if } y_i \text{ is positive but we say negative} \\ w_{i-1} - x_i & \text{if } y_i \text{ is negative but we say positive} \end{cases}$$



Thm: the perceptron makes at most $\frac{1}{\gamma^2} + 1$ mistakes ($\gamma \equiv$ margin of w_*)

Proof: When we make a mistake $w_i = w_{i-1} + y_i \cdot x_i$ we have $\langle w_i, w_* \rangle = \langle w_{i-1}, w_* \rangle + y_i \langle x_i, w_* \rangle$

$\Rightarrow \langle w_i, w_* \rangle = \langle w_{i-1}, w_* \rangle + |\langle w_*, w_i \rangle| \Rightarrow \geq \langle w_{i-1}, w_* \rangle + \gamma$ because $\gamma = \frac{\langle w_*, x \rangle}{\|w_*\| \cdot \|x\|}$. For every mistake i we have that $\langle w_{i-1}, w_* \rangle + \gamma \leq \langle w_i, w_* \rangle$. We are improving the inner product.

$\|w_i\|^2 = \|w_{i-1} + y_i x_i\|_2^2 = \|w_{i-1}\|^2 + \|y_i \cdot x_i\|_2^2 + 2\langle w_{i-1}, y_i x_i \rangle = \|w_{i-1}\|_2^2 + 2y_i \langle w_{i-1}, x_i \rangle$. Since this only happens when we made a mistake we have that $\|w_i\|_2^2 \leq \|w_{i-1}\|_2^2 + 1$.

→ Whenever we make a mistake we have that $\langle w_{i-1}, w_* \rangle + \gamma \leq \langle w_i, w_* \rangle$ and $\|w_i\|^2 \leq \|w_{i-1}\|^2 + 1$. At any time T, if we made M mistakes then $\|w_T\|^2 \leq 1 + M$. so we have: $\langle w_0, w_* \rangle + M\gamma \leq \langle w_T, w_* \rangle \leq \|w_T\| \cdot \|w_*\|$

$M \cdot \gamma \leq \sqrt{1+M} - \langle w_0, w_* \rangle \leq \sqrt{1+M} - 1$. So we have that $M \cdot \gamma \leq \sqrt{1+M} - 1$. By some basic calculation we have that $M \leq \frac{1}{\gamma^2} + 1$.

Refer to Apendix D for more detailed calculations

Online Learning and SGD:

Stochastic gradient descent is essentially already an online algorithm: Perceptron ≡ SGD with hinge loss recall that if $\ell(a,b)$ is hinge loss then: $\nabla\ell(\langle w_{i-1}, x_i\rangle, y_i) = 0$ if they are same sign, otherwise $\nabla\ell = y_i \cdot x_i$

Regret Minimization:

→ Sometimes called "learning with experts"

Day	E_1	E_2	E_3	.....	E_d	Our	Truth
1	UP	↓	↑		Up	Up	Up
Score	0	1	0		0	0	
2	UP	UP	Down		UP	↓	↓
Score	1	1	0		1	0	

The example on the right has us pick an expert to follow and we then we follow that expert. We have the ability to switch experts if an expert does better or worse

→ Eg: Stock Market Prediction with "d" experts

$L(t,i) \equiv$ loss incurred by expert i on day t.

$L(t) \equiv$ loss incurred by our algorithm on day t → we want to figure out the best expert

$$\text{Regret}(T) = \sum_{t=1}^{T} L(t) - \min_i \sum_{t=1}^{T} L(t,i)$$

our-less; loss of best expert

→ our goal is to minimize the regret.

→ Follow the leader strategy: the regret of following the leader is the loss of best expert T/d. While our loss would be T. The formula would be $(1-\frac{1}{d})T$

→ Follow the majority: we maintain a set of experts, when someone makes a mistake throw them out. This will work when we have an infalable expert.

Follow the Majority (FTM):

- $d[0] = \{1,2,\ldots,d\}$
- On each day $t = 1,\ldots,d$:
  - → Predict according to the majority prediction of experts in $d(t-1)$
  - → $d(t) = d(t-1) \setminus$ experts who made mistake

Thm: $\text{Regret}(T) \le \log_2(d)$

Proof: whenever we make a mistake, the set of experts shrinks by at-least a factor of 1/2.

Potential Function (tracks progress of algorithm): $W(t) = |d(t)|$ ← # of experts

$W(0) = d$ and $W(T) \le \frac{W(t-1)}{2}$. Let $L(t)$ = # mistakes made by algorithm until time step t. Since there is at least one-infalable expert: $1 \le W(t) \le d \cdot (\frac{1}{2})^{L(t)}$. Now we have $1 \le d \cdot (\frac{1}{2})^{L(t)} \Rightarrow 2^{L(t)} \le d$ which implies $L(t) \le \log_2 d$. This means $\text{Regret}(FTM) \le \log_2(d)$

The assumption of having an infallable expert is too strong so instead we look to give a "weight" to each expert and reduce/increase the weights or confidence for the experts.

Weighted Majority:

→ We make predictions based on the weighted majority.

→ Every mistake will "half" your weight (This can be a different constant)

Follow the Weighted Majority (FTWM)

- $W(0,i) = 1 \quad \forall\ i=1,\dots,d$

On each day $t=1,\dots,T$

→ Predict according to weighted majority with weights $w(t-1,i)$

→ For each expert i:

If mistake: $W(t,i) = \frac{W(t-1,i)}{2}$

Thm: $L(T) \le 2.4(L_*(T) + \log_2 d)$

($L(T)$: our loss; $L_*(T)$: loss of best-expert)

Proof:

$W(t) = \sum_{i=1}^{d} W(t,i) = \sum_{c} W(t-1,c) + \frac{1}{2}\sum_{i} W(t-1,i)$. By weighted majority rule we have that $\frac{1}{2}\sum_{i} W(t-1,i)$ is greater than $\frac{W(t-1)}{2}$. So we have that $W(t) \le \frac{W(t-1)}{2} + \frac{1}{2}\cdot\frac{W(t-1)}{2} \Rightarrow W(t) \le \frac{3}{4}W(t-1)$.

If $L(t)$ is the total number of mistakes we have made after t-days. Hence $\boxed{W(t) \le d\cdot\left(\frac{3}{4}\right)^{L(t)}}$

What about our lower bound. Hence we have that if the the best expert has made $L_*(t)$ mistakes then we have that $\left(\frac{1}{2}\right)^{L_*(t)} \le W(t)$

→ $\left(\frac{1}{2}\right)^{L_*(t)} \le W(t) \le d\cdot\left(\frac{3}{4}\right)^{L(t)} \Rightarrow \left(\frac{4}{3}\right)^{L(t)} \le d\cdot 2^{L_*(t)}$

→ $L(t) \le \frac{1}{\log_2(4/3)}\left(L_*(t) + \log_2 d\right)$

→ $L(t) \simeq 2.4\left(L_*(t) + \log_2 d\right)$ ■

$\text{Regret}_{FTWM}(T) = L(T) - L_*(T) = 1.4\,L_*(T) + 2.4\log_2(d)$

However supposse that the best expert is right 90% of the time. Then $L_*(T) \approx 0.1(T)$ hence: Regret(T) is increasing. So we may not want to just track regret(T) but how our regret relative to the number of times we have played the game: $\boxed{\frac{\text{Regret}(T)}{T}}$. As $T \to \infty$ we want $\text{Regret}(T)/T \to 0$. These are called "No regret algorithms."

FTWM is not a no-regret algorithm.

Claim: There is no deterministic algorithm for "no-regret". There is no deterministic algorithm that can do better than a factor of 2.

There exist non-deterministic algorithms (random algorithms) with a certain expectation will have our regret go to zero. One such method is the multiplicative weights method (MWM) which is covered on the next page.

Multiplicative Weights Method (MWM):

$w(0,i) = 1, \ \forall i = 1,\dots,d$

On each day $t = 1,\dots,T$

Pick an expert i with probability proportional $w(t-1,i)$

$\Pr[\text{expert } i \text{ is picked}] = \dfrac{w(t-1,i)}{\sum_{i=1}^{d} w(t-1,i)}$

Then follow expert i

Update weights of every expert:

If wrong: $w(t,j) = (1-\varepsilon)w(t-1,j)$

Remark: Prediction ≡ Add up weights of up expert add up weights of down experts. Predict UP with probability proportional to the total weight that predicted up.

Thm: $\mathbb{E}[L(T)] \le (1+\varepsilon)L_*(T) + \dfrac{\ln(d)}{\varepsilon}$ when $\varepsilon < 1/2$.

Proof: Track the total weight: $W(t) = \sum_{i=1}^{d} w(t,i)$

$$\mathbb{E}[\text{loss incurred on day } t] = \sum_{i=1}^{d} \Pr[\text{We pick expert } i] \cdot L(t,i)$$

$$= \sum_{i=1}^{d} \frac{w(t-1,i)}{\sum_{j=1} w(t-1,j)} \cdot L(t,i)$$

$$\mathbb{E}[\text{Loss on day } t] = \sum_{i=1}^{d} \frac{w(t-1,i)}{W(t-1)} \cdot L(t,i)$$

$$= \frac{1}{W(t-1)} \sum_{i=1}^{d} w(t-1,i) \cdot L(t,i) \quad (1)$$

Recall that: $w(t) = \sum_{i=1}^{d} w(t-1,i)\,(1-\varepsilon \cdot L(t,i)) = \sum_{i=1}^{d} W(t-1,i) - \varepsilon \cdot \sum_{i=1}^{d} w(t-1,i) L(t,i)$

$$\Rightarrow W(t) = W(t-1) - \varepsilon \sum_{i=1}^{d} w(t-1,i) L(t,i) \quad (2)$$

Combining 1 and 2 we get $W(T) = W(t-1) - \varepsilon \cdot W(t-1) \cdot \mathbb{E}[\text{loss on day } t]$

$$\Rightarrow W(t) = (1 - \varepsilon \cdot \mathbb{E}(\text{loss}))\, W(t-1) \quad (3)$$

New idea: compute an upper and lower bound of $W(t)$. For all $1-\alpha \le e^{-\alpha} \ \forall \alpha$

Hence we have that:

$$W(t) \le (1-\varepsilon \mathbb{E}(t))W(t-1) \le W(t-1)e^{-\varepsilon \mathbb{E}(t)}$$

$$W(1) \le W(0) \cdot e^{-\varepsilon \mathbb{E}(1)}$$

$$W(2) \le W(1) \cdot e^{-\varepsilon \mathbb{E}(2)}$$

So by telescoping we have:

$$W(t) \le e^{-\varepsilon \mathbb{E}(1)} \cdot e^{-\varepsilon \mathbb{E}(2)} \cdot \dots \cdot e^{-\varepsilon \mathbb{E}(t-1)} \cdot e^{-\varepsilon \mathbb{E}(t)} \cdot W(0)$$

$$\Rightarrow W(t) \le e^{-\varepsilon \sum_{j=1}^{t} \mathbb{E}(t)} \cdot W(0) \Rightarrow W(t) \le e^{-\varepsilon L(t)} \cdot d \quad (4)$$

Our best loss would be the loss of the best expert: $W(t) \ge (1-\varepsilon)^{L_*(t)}$ (5)

Combining 4) and 5) we get that: $L(t) \le \dfrac{\ln(\frac{1}{1-\varepsilon})}{\varepsilon} L_*(t) + \dfrac{\ln(d)}{\varepsilon}$ ■

Corollary: if we set $\epsilon = \sqrt{\frac{\ln(d)}{T}}$ then we get $\mathbb{E}[L(T)] \leq L_*(T) + 2\cdot\sqrt{T\cdot\ln(d)}$

The proof of corollary:

$$\mathbb{E}[L(T)] \leq L_*(T) + \sqrt{\frac{\ln(d)}{T}}\cdot L_*(T) + \frac{\ln(d)\sqrt{T}}{\ln(d)}$$

$\Rightarrow$ If we run for T rounds, the biggest loss we can get would be being wrong each time so we get the largest loss of $L_*(T)$ to be T.

$\Rightarrow L_*(T) + \sqrt{T\cdot\ln(d)} + \sqrt{T\cdot\ln(d)} = L_*(T) + 2\cdot\sqrt{T\cdot\ln(d)}$ ■

Corollary: the Regret of MWM is a no-regret algorithm

Proof: $\text{Regret}_{MWM}(T) = \mathbb{E}[L(T)] - L_*(T)$

$$\Rightarrow \text{Regret}_{MWM}(T) \leq 2\cdot\sqrt{T\cdot\ln(d)}$$

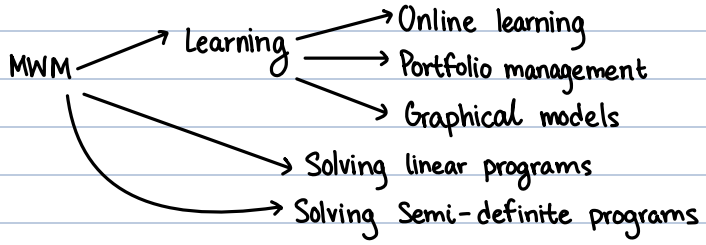
$$\Rightarrow \frac{\text{Regret}_{MWM}(T)}{T} \leq 2\cdot\sqrt{\frac{\ln(d)}{T}}$$ we see that as $t\to\infty$ we have that $\frac{\text{Regret}_{MWM}(T)}{T} = 0$ ■

Remark: $\Omega(\sqrt{T})$ is the best possible regret

$\Omega(\log d)$ is also the best possible bound for regret

While there are specalized cases where we can do better, in-general the above hold true.

Applications of MWM:



MWM is a very important algorithmic tool that has a wide variety of applications that go beyond online learning.

Boosting:

Goal: we want 90% accuracy, and suppose we have some classifier that gives 60% accuracy. The big question is can we boost the accuracy of a "weak learner" to a "strong learner". Without tinkering with weak-learner

→ Weak learner: slightly better than random guessing

→ Strong learner: slightly worse than 100% accuracy.

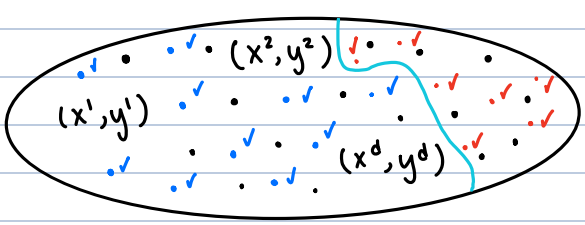
Boosting on Samples:

Dataset: $(x^1,y^1),(x^2,y^2),\ldots,(x^d,y^d) \in \mathcal{X}\times d = \{0,1\}$

Hypothesis Class H:

Weak learner assumption: every distribution D on the dataset, we can find a $h\in H$ such that $\Pr[h(x^i)\neq y_i]$ will be $\leq \gamma$. (Say $\gamma = 0.4$)

Goal: build a stronger learner to get error $\leq \delta$ (say $\delta = 0.1$).



→ Run WL (Weak learner) on entire data-set

→ Step 1: Put "more weight" on points that were wrong

→ Repeat Process.

A framework for Boosting:

→ Start with $D^{(0)} = (1/d, 1/d, \dots, 1/d)$, a distribution of d-weights that all add up to one and are equal.

→ $h^{(0)} = WL(D^{(0)})$

- For $t=1,\dots,T$:
  - → Update $D^{(t-1)}$ to $D^{(t)}$
  - → $h^{(t)} = WL(D^{(t)})$
- Output $h_{strong} = Combine(h^{(0)},\dots,h^{(T)})$

The two black boxes we are going to deal with are:

Update & Combine

Defining Combine function:

Natural Idea: Combine ≡ Majority.

$Majority(h^{(0)}, h^{(1)}, \dots, h^{(T)})(x) = Majority(h^{(0)}(x), \dots, h^{(T)}(x))$

Definining the Update:

Idea use MWM to update the distribution $D^{(t-1)} \to D^{(t)}$

Imaginar learning w. experts game:

	(x^1,y^1)	(x^2,y^2)		(x^d,y^d)
h^0	✓	x	✓x✓....	x
Loss	1	0	101....	0
h^1	x	✓	x✓✓	✓
Loss	0	1	011	1

Adaboost:

- $D^{(0)} = (\frac{1}{d}, \dots, \frac{1}{d}) \cdot w(0,i) = 1$
- $h^{(0)} = WL(D^{(0)})$, $i=1,\dots,d$
- For $t=1,\dots,T$:
  - → Define:

$$w(t,i) = \begin{cases} (1-\varepsilon)w(t-1,i) & \text{if correct} \\ w(t-1,i) & \text{if wrong} \end{cases}$$

  - → $D^{(t)}$ is distribution whose probabilities are proportional to weights
  - → $h^{(t)} = WL(D^{(t)})$
- $h_{strong} = Majority(h^{(0)}, h^{(1)}, \dots, h^{(T)})$.

Thm: Adaboost achieves accuracy $1-\delta$ on the dataset if $T \geq \frac{2\ln(1/\delta)}{(\frac{1}{2}-\gamma)^2}$

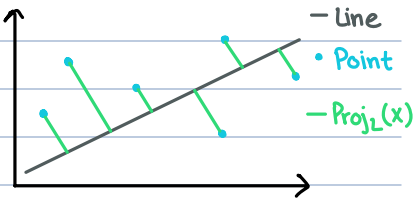
Principle Component Analysis (PCA):

What happens when we have a-lot of unsupervised data what can our learning goals be? We could try and group similar items together, attempt to identify outliers in the data,

PCA is an unsupervised technique, we take the following steps: we identify the direction in which the data varies the most. This is the first principle component, look for a second feature

The goal of PCA is to reduce the dimension and then apply classical learning methods. PCA can still be used and still often used in supervised learning.

Best Fit Line Problem:



→ Input: $x^1, x^2, \dots, x^n \in \mathbb{R}^d$

→ Output: find the line that is closest to the dataset

Defining Closest: minimize aggregate distance of points to the line.

Given a line L, and a point x, $\text{Proj}_L(x)$ = closest point on L to x. Note that $\text{Proj}_L(x)$ = the point on L that minimizes $\|x-y\|_2$.

Our new problem is: Find an L that minimizes aggregate distance $\|x-\text{Proj}_L(x)\|$. So we have

$$\text{ERR}(L,x) = \sum_{i=1}^{n} \|x_i - \text{Proj}_L(x_i)\|_2^2.$$

General Best Fit Subspace Problem (BFS):

Input: $x^1, x^2, \dots, x^n \in \mathbb{R}^d$, and an integer integer $k \geq 1$. For any k-dimensional subspace $S \subseteq \mathbb{R}^d$ we define $\text{ERR}(S, x) = \sum_{i=1}^{n} \|x^i - \text{Proj}_S(x^i)\|_2^2$. (A line is like a 1-dimensional subspace)

Output: Find a k-dim subspace S that minimizes $\text{ERR}(S_i, x)$, let us return k-orthonormal vectors that form a basis for S.

Designing the algorithm:

We have an input $x^1, x^2, \dots, x^n \in \mathbb{R}^d$. We want to find the line that minimizes $\text{ERR}(L,x) = \sum_{i=1}^{n} \|x^i - \text{Proj}_L(x^i)\|^2$

By pythagorean theorm (Appendix A, Projections, #2): $\sum_{i=1}^{n} \|x^i\|^2 - \sum_{i=1}^{n} \|\text{Proj}_L(x^i)\|^2$

↑ This is constant! ↘ Since we are subtracting we max

To minimize the above is the same as maximizing $\sum_{i=1}^{n} \|\text{Proj}_L(x^i)\|^2$. Let us denote this as $\text{Var}(L, x)$

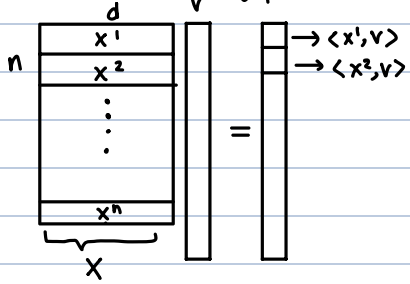
Hence we have that minimizing $\text{ERR}(S, x) \equiv$ Maximizing $\text{Var}(S, x)$

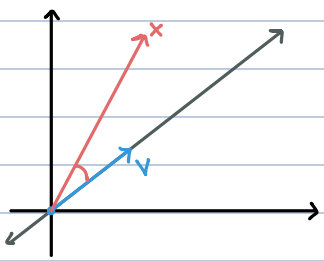
Maximizing Variance for dim 1:

Given a vector v for dimension 1 (a line) is $S = \text{span}(\{v\})$

$$\text{Var}(L,x) = \sum_{i=1}^{n} \|\text{Proj}_L(x)\|^2 = \sum_{i=1}^{n} \frac{\langle x^i, v\rangle^2}{\|v\|^2} \Rightarrow \text{Var}(S,v) = \frac{\|X \cdot v\|^2}{\|v\|^2}$$

Where:





$$\|\text{Proj}_L(x)\| = \left\langle x, \frac{v}{\|v\|_2} \right\rangle$$

So now we have that the best fit line problem is equivalent to: find v that maximizes $\frac{\|X \cdot v\|^2}{\|v\|^2}$. The unit norm solution of this problem is called First Principle component of X.

Example #1:

$$X = \begin{bmatrix} -10 & 0 & 0 & \cdots & 0 \\ -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ -10 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ -11 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

The first PC of X: $[1, 0, 0, \ldots, 0]$

Example #2:

$$X = \begin{bmatrix} u \\ 10u \\ -11u \\ \vdots \\ -10u \end{bmatrix}$$

Where u is a unit vector. So we have that the first PC of X: u

Moving onto higher dimensions:

What about $k=2$?

Idea #1:

1). Find first PC $\rightarrow v'$

2). Replace each x^i with $\bar{x}^i = x^i - \text{Proj}_{\{v\}}(x^i)$

3). Find first PC of the new dataset $\{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$

Idea #2:

1). Find first PC v'

2). Find direction max $\text{Var}(S, v)$ but $v \perp v'$.

3). Find $v^2 \equiv \underset{v \perp v'}{\text{argmax}} \frac{\|x \cdot v\|^2}{\|v\|^2}$

The second right singular vector or second PC of $X \equiv \underset{v: \|v\|=1, \langle v, v' \rangle = 0}{\text{argmax}} \|X \cdot v\|^2$. It turns out idea #1 and #2 are the same, but #1 is easier computationally.

If $v', v^2, \ldots, v^{i-1}$ are the first, second, ..., $(i-1)$ right singular vectors, then:

$\underset{v: \|v\|=1, v \perp v', v \perp v^2, \ldots v \perp v^{i-1}}{\text{argmax}} \|X \cdot v\|^2$ is the i^{th} right singular vector

Thm: The span of the first k right singular vectors maximizes $\text{Var}(S, x)$ and therefore minimizes $\text{ERR}(S, x)$ and solves the best fit subspace problem.

Remark: if we defined $\overline{\text{ERR}}(S, x) = \sum_{i=1}^{n} \|x^i - \text{Proj}_S(x^i)\|$, then finding the best fit line, and another line is not equivilent to solving the best fit subspace problem. The power of 2 is what allows for us to make the guarantee that we get the optimal solution.

Thm #1: Span of first two right singular vectors solves the best fit line problem:

Proof:

$v^1 \longrightarrow$ first singular vector

$v^2 \longrightarrow$ second singular vector

and $S = \text{span}(\{v^1, v^2\})$ we want to show $\text{Var}(S, x)$ is maximized given $\text{Var}(S,x) = \sum_{i=1}^{n} \|\text{Proj}_S(x^i)\|_2^2$

Linear Algebra Fact: $\|\text{Proj}_T(x)\|^2 = \langle x, w^1\rangle^2 + \langle x, w^2\rangle^2$ since $w^1 \perp w^2$.

$\Rightarrow \text{Var}(S,x) = \sum_{i=1}^{n} \langle x^i, v^1\rangle^2 + \langle x^i, v^2\rangle^2 = \|X \cdot v^1\|^2 + \|X \cdot v^2\|^2$

↳ We want to show this the maximum score we can get.

Take any other 2-dimensional subspace T with a basis w^1, w^2.

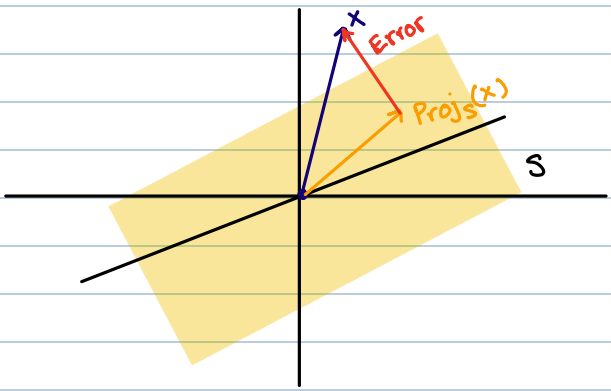
$\text{Var}(T,x) = \sum_{i=1}^{n} \|\text{Proj}_T(x^i)\|^2 = \|X \cdot w^1\|^2 + \|X \cdot w^2\|^2$

From definition we have that v^1 is greater than w^1 since v^1 maximizes the length of $\|X \cdot v^1\|^2$. But we cannot say that directly for v^2 and w^2 since by definition we have that v^2 maximizes the length of $\|X \cdot v^2\|^2$ for *all* vectors perpendicular to v^1 so, there exists the possibility that $w^2 = v^1$ which means $w^2 > v^2$.

However $\exists$ an orthonormal basis w^1, w^2 for T where $w^2 \perp v^1$. Then $\|X \cdot w^1\|^2 \leq \|X \cdot v^1\|^2$ by definition of first PC and $\|X \cdot w^2\|^2 \leq \|X \cdot v^2\|^2$ by definition of second PC. Then we get that $\text{Var}(S,x) \geq \text{Var}(T,x)$. Hence it is proven that

---

Linear Algebra Facts:



1) $\text{Proj}_S(x) = \arg\min_{y \in S} \|x - y\|_2$

2) Angle between $x - \text{Proj}_S(x)$ and $\text{Proj}_S(x)$ is 90°. Thus their dot product is zero. $\langle x - \text{Proj}_S(x), \text{Proj}_x(S)\rangle = 0$

3) $\|x\|^2 = \|\text{Proj}_S(x)\|^2 + \|x - \text{Proj}_S(x)\|^2$ (↑ error)

4) $\forall u \in S$ we have that it will be perpendicular to the error vector: $\langle u, x - \text{Proj}_S(x)\rangle = 0$

5) $\|\text{Proj}_L(x)\| = \langle x, \frac{v}{\|v\|}\rangle$ where L is a line and v is a vector on that line.

6) Cauchy-Shwartz inequality: $\langle u, v\rangle \leq \|u\| \cdot \|v\|$ we also have $2\langle u, v\rangle = \|u\|^2 + \|v\|^2 - \|u - v\|^2$

7) The dot product of two orthonormal vectors v_i and v_j is $\langle v_i, v_j\rangle = 0$ if $i \neq j$ and $\langle v_i, v_j\rangle = 1$ if $i = j$

8) If the matrix X has orthogonal rows or cols then $X^T \cdot X = I_n$ and $X \cdot X^T = I_n$

9) A similar matrix is defined as when $X = X^T$.

timer10) L_1-norm: $\|x\|_1 = \sum |x_i|$

Low Rank Approximation:

We are given some $n \times d$ matrix and our goal is to approximate X by a "simple matrix".

We must answer the following questions:

1). Meaning of simple? $\longrightarrow$ low rank

2). What does approximate mean?

Recall: Rank(x) = dim(span of cols) = dim(span of rows)

A matrix X has rank r only if $X = A \cdot B$ where $A = n \times r$ and $B = r \times d$.

Why do we care about rank?

$\longrightarrow$ too store an $n \times d$ it takes $n \cdot d$ space to store

$\longrightarrow$ too store an $n \times d$ with rank (r) space to strore is $n \cdot r + d \cdot r = (n+d)r$

$\longrightarrow$ matrix multiplication: $n \cdot d$ time

$\longrightarrow$ matrix multiplication: $(n \cdot r) + (r \cdot d) = (n+d)r$ time

What about approximation:

Squared Error (RMSE): $\left(\sum_{i=1}^{n} \sum_{j=1}^{d} (x - \tilde{x})_{ij}^2 \right)^{1/2} = \|x - \tilde{x}\|_F$

For any matrix the frobenius norm: $\|A\|_F = \left(\sum_{i,j} A_{ij}^2 \right)^{1/2}$

Low-Rank Approximation Problem:

Input: $x \in \mathbb{R}^{n \times d}$ and some k

Output: $\tilde{x}$ to minimize $\|x - \tilde{x}\|_F$ of $\text{rank}(\tilde{x}) \leq k$.

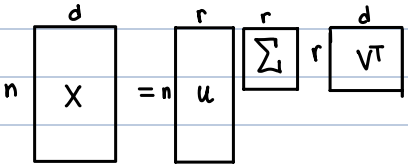
The Netflix Challenge:

$\longrightarrow$ Provided 500k users and users ratings of 17k different movies (users didn't rate all 17k) Goal is to complete the matrix, there are empty ratings which would be helpful to be able to recomend through their recommendation system.

$\longrightarrow$ Applying low-rank approximation beat the Netflix teams baseline by 4%

Singular Value decomposition:

Thm: Any matrix can be written as a product of 3 matricies: $X = U \Sigma V^T$ where the following properties are true:



1.) Columns of U are orthonormal $(U^T U = I_r)$

2.) Columns of V are orthonormal $(V^T U = I_r)$

3.) Σ is a diagonal matrix with non-negative entries

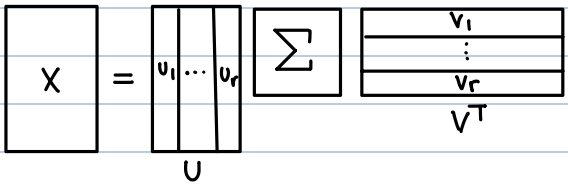
Ex: $\begin{bmatrix} 1 & 0 \\ 2 & 3 \\ 0 & 4 \\ 5 & \end{bmatrix} = \mathbb{I} \cdot \begin{bmatrix} 1 & 0 \\ 2 & 3 \\ 0 & 4 \\ 5 & \end{bmatrix} \cdot \mathbb{I}$

Ex: $\begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Ex: Say X is an orthonormal matrix: $X = X \cdot I \cdot I$

Magical Properties of SVD:

→ We can permute our matricies such that $\Sigma_{11} \geq \Sigma_{22} \geq \Sigma_{33} \geq \ldots \geq \Sigma_{rr}$



Thm: v_i is the i^{th} principal component of X

Thm: taking first k-cols & first k-rows is a solution to best rank(k) approximation

SVD Properties:

→ Assume we always write it as a decomposition where $\Sigma_{11} \geq \Sigma_{22} \geq \ldots \geq \Sigma_{rr}$

1). We can express it as the following: $X = \sigma_1 \cdot u_1 \cdot v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_r u_r v_r^T$

2.) $X \cdot v_i = \sigma_i \cdot u_i$

Proof: $(\sigma_1 u_1 v_1^T + \ldots + \sigma_r u_r v_r^T) \cdot v_i$ using orthonormality $\Rightarrow \sigma_1 u_1 (v_1^T \cdot v_i) + \ldots + \sigma_r u_r (v_r^T \cdot v_i)$ and $v_j^T \cdot v_i = 0$ if $i \neq j$ and 1 else hence we have $\sigma_i u_i$

3.) $u_i^T X = \sigma_i \cdot v_i$ (Same proof as above)

4.) $\sigma_i = \|X \cdot v_i\|$ since $\|u_i\|$ is 1 since u_i is a unit vector

5.) $u_i = \frac{X \cdot v_i}{\|X \cdot v_i\|}$ by combining 2 and 4

If we know the v's we can infer σ and u_i

Thm: We can compute the full SVD in time $O(n \cdot d^2)$

For very large data this can still be very expensive to compute, so is there a way to accurately compute first k-singular vectors.

Power Iteration:

→ can approximate first few singular vectors very quickly:

Note: $X = U \Sigma V^T = \sigma_1 u_1 v_1^T + \ldots + \sigma_r u_r v_r^T$

let $Y = X^T \cdot X = (U \Sigma V^T)^T \cdot (U \Sigma V^T) = V \Sigma U^T \cdot U \cdot \Sigma V^T = V \Sigma^2 V^T$; Y is a d×d matrix

let $Y^2 = Y \cdot Y = V \Sigma^4 V^T$

let $Y^3 = V \Sigma^6 V^T$

then $Y^l = V \Sigma^{2l} V^T$ note that Σ^{2l} is still diagonal with entries σ_i^{2l} as it's entries

$Y^l = V \Sigma^{2l} V^T = \sigma_1^{2l} v_1 v_1^T + \sigma_2^{2l} v_2 \cdot v_2^T + \ldots + \sigma_r^{2l} \cdot v_r v_r^T$ we observe that since $\sigma_1 \geq \sigma_r$ that σ_r must decay exponentially compared to σ_1 hence as $l \to \infty$ $Y^l \approx v_1 \cdot v_1^T$. If we look at the first column of Y^l and normalize we will get v_1.

Idea:

1) Compute Y^l for large l
2) Compute first column of $Y^l \Rightarrow \bar{v} = Y^l \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ with this we avoid matrix×matrix multiplication.
3) Output $\frac{\bar{v}}{\|\bar{v}\|}$

Analysis of Power Iteration:

Input: $X \in \mathbb{R}^{n \times d}$ where $n > d$

Output: v s.t. $\max_{\|v\|=1} \|X \cdot v\|$

1.) $\bar{V}_0$ = random unit vector

2.) For $t = 1, \ldots, \ell$:

$\bar{V}_t = X^T(X \cdot \bar{V}_{t-1})$

3.) Output $\frac{\bar{V}_\ell}{\|\bar{V}_\ell\|}$

Note we may run into precision issues since our values grow quadratically we can avoid this by normalizing after every iteration.

Thm: PI will converge to a top right singular vector after $T = O\left(\frac{\log(\frac{d}{\epsilon})}{\epsilon}\right)$ iterations and $\|X \cdot \bar{V}_T\| \geq (1-\epsilon)\sigma_1$

Proof:

Let $X = U \sum V^T$ be it's SVD with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$ let σ_m be the vector s.t. $\sigma_m \geq (1-k)\sigma_1$ and then $\sigma_{m+1} < (1-\epsilon)\sigma_1$

Let S_m = span of the first m singular vectors $v^1, v^2, v^3, \ldots, v^m$, and as long as our vector is within this span we will be good. We can compare how much of the vector is in the span by using a projection: $\|\bar{V}_T - \text{Proj}_{S_m}(\bar{V}_T)\| \leq \delta$

We first show after T iterations: $\|\bar{V}_T - \text{Proj}_{S_m}(\bar{V}_t)\| \leq \frac{(1-\epsilon)^{2T}}{|\langle V_0, v^1 \rangle|}$ for a random vector V_0 then $|\langle V_0, v^1 \rangle| \approx 1/\sqrt{d}$

Suppose that $\sigma_2 < (1-\epsilon)\sigma_1$ then writing the power iteration: $\bar{V}_t = \frac{(X^T \cdot X)^t \cdot V_0}{\|(X^T X)^t \cdot V_0\|}$

$X^T X = \sum_{i=1}^{r} \sigma_i v_i v_i^T \Rightarrow (X^T X)^T = \sum_{i=1}^{r} \sigma_i^{2t} v_i v_i^T$ since $v^1, \ldots, v^r$ are orthonormal we extend it to create a basis: $\{v_1, \ldots, v_r, v_{r+1}, \ldots, v_d\}$. We then express V_0 in terms of this basis:

$V_0 = c_1 V_1 + c_2 V_2 + \ldots + c_d V_d$ hence $(X^T X)^T V_0 = \sum_{i=1}^{r} \sigma_i^{2t} v_i v_i^T (c_1 V_1 + \ldots + c_d V_d)$

by orthonormality we have: $\sum_{i=1}^{r} \sigma_i^{2t} \cdot c_i \cdot v_i$

We now compute: $\|\bar{V}_T - \text{Proj}_{\text{span}(v^1)}(\bar{V}_t)\|$ by definition: $\text{Proj}_{\text{span}(v_1)}(\bar{V}_t) = \frac{1}{\|(X^T X)^T \cdot V_0\|}(\sigma_1^{2t} \cdot c_1)(v_1)$

thus $\|\bar{V}_T - \text{Proj}_{\text{span}(v_1)}(\bar{V}_t)\| = \frac{1}{\|(X^T X)^T V_0\|^2} \cdot \| (X^T X)V_0 - \text{Proj}_{\text{span}(v_1)}((X^T X)^T V_0)) \|$

$= \frac{1}{\|(X^T X)^T V_0\|^2} \sum_{i=2}^{r} \sigma_i^{4t} c_i^2$ since if $(v_1, \ldots, v_d)$ the error of projecting onto the first coordinate: $v_2^2 + \ldots + v_d^2$

$\leq \sum_{i=2}^{r} (1-\epsilon)^{4t} \sigma_1^{4t} c_i^2$

$= (1-\epsilon)^{4t} \sigma_1^{4t} \sum_{i=2}^{r} c_i^2 \leq (1-\epsilon)^{4t} \cdot \sigma_1^{4t}$

The denominator $\|(X^T X)^T V_0\|^2 = \| \sum_{i=1}^{r} \sigma_i^{2t} c_i v_i \|^2 = \sum_{i=1}^{r} \sigma_i^{4t} c_i^2 \geq \sigma_1^{4t} \cdot c_1^2$

Hence the ratio: $\leq \frac{(1-\epsilon)^{4t} \sigma_1^{4t}}{c_1^2 \sigma_1^{4t}} \leq \frac{(1-\epsilon)^{2T}}{c_1^2}$ note that c_1^2 is $\langle V_0, V_1 \rangle$ hence: $\leq \frac{(1-\epsilon)^{2T}}{\langle V_0, V_1 \rangle}$

Plugging in the proper value therefore results in the proof of the thm.

Computing i^{th} singular vector

Remark: Suppose $X = X_1 \cdot X_2 \cdot X_3$ we want to avoid matrix multiplication so in the PI step we do:
$(X_1X_2X_3)^T(X_1 \cdot X_2 \cdot X_3)(V_t) = (X_3^T \cdot X_2^T \cdot X_1^T)(X_1X_2X_3)(V_t) = X_3^T(X_2^T(X_1^T(X_1(X_2(X_3 \cdot V_t)))))$

Computing 2^{nd} singular vector:
$X = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_r u_2 v_2^T$
Compute 1^{st} singular vector: $X_1 = X - \sigma_1 u_1 v_1^T$
Computing 3^{rd} singular vector:
Compute 1^{st} singular vector: $X_2 = X_1 - \sigma_2 u_2 v_2^T$

However with this method results in a matrix loosing sparsity however we can instead note that to compute $X_1 \cdot v = X \cdot v - \sigma_1 u_1 (v_1^T \cdot v)$ instead so that we can still avoid matrix-matrix multiplication.

General Power Iteration:

→ Pick k orthonormal vectors $v_{0,1}, v_{0,2}, \ldots, v_{0,k}$
→ For $t = 1, \ldots, T$:
- $u_{t+1} = (X^T(X(v_{t-1,1}))$
- $u_{t,2} = (X^T(X(v_{t-1,2}))$
- $u_{t,k} = (X^T(X(v_{t-1,k}))$
- Let $v_{t,1}, \ldots, v_{t,k}$ be an orthonormal basis for span$(u_{t+1}, u_{t,2}, \ldots, u_{t,k})$

We are running k-power iterations simaltaneously
We can also create power iteration + momentum which would have an even faster convergence
PI has convergence : $1/\varepsilon$
PI has momentum : $1/\sqrt{\varepsilon}$
The convergence of PI depends on the relative gap between $\sigma_1, \sigma_2, \ldots, \sigma_r$

Matrix Completion & Singular Value Projection

Input: we are given some entries where some are missing: let $\Theta \subseteq [n] \times [d]$ and we have access to entries x
Output: find $\hat{x}$ of rank $\leq k$ such that observed error is minimized $\min \sum_{i,j \in \Theta} (x_{ij} - \hat{x}_{ji})^2$

Take a loss function $L(Y) = \sum_{(i,j) \in \Theta} (x_{ij} - Y_{ij})^2$ however this is a constrained optimization problem since we have a rank limit. Instead if we step outside we project back onto the right set: $\text{Proj}(Y_t') = \underset{z : \text{rank}(z) \leq k}{\arg\min} \|z - Y_t'\|_F^2$ which can be done by using SVD.

Singular Value Projection (SVP)
→ Start with matrix Y_0:
→ For $t = 1, \ldots, T$:
$y_t' = Y_{T-1} - \eta \nabla L(Y_{t-1})$ (∇L is zero outside Θ)
Y_t = best rank-k approximation to y_t' by SVD

*This does very well for sparse matricies and we note that $Y_t' \equiv$ low rank − sparse matrix

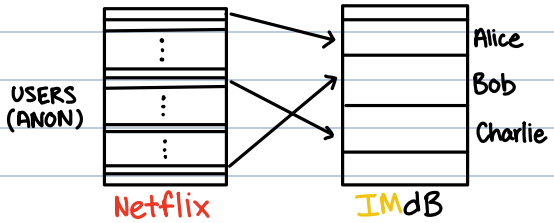
*Matrix Completion is NP-hard, but this can be used to do fairly well approximations.

Privacy:

There are many sensitive datasets:

→ Healthcare data, genetic data, search history, text history are all sensitive data and we would like to keep this information obfuscated so that a particular datapoint does not get matched to a particular user.

Ex: In the Netflix Challenge the netflix data was private and anonymous but users could be tied back and matched back to their imdb reviews:



We can have correlation attacks, and even reconstruction attacks. (The above was a correlation attack)

How to get Privacy?

→ Suppose we just want to calculate the mean or average of a single entry dataset, how do we ensure privacy? We will have no privacy if we release all the data, and we get complete privacy if we release only random data what is the middle ground?

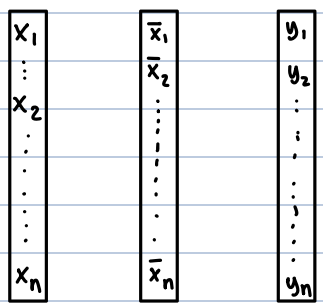
→ "Randomized Response": we collect a response and then keep it or invert it out with a probability P

Randomized Response:

$$\overline{X_i} = \begin{cases} X_i & \text{with prob } 1/2+\gamma \\ 1-X_i & \text{with prob } 1/2-\gamma \end{cases}$$

where $0 \le \gamma \le 1/2$ we can vary gamma to get a different level of privacy. $Y_i \in \{0,1\}$ and $X_i \in \{0,1\}$ either 0 or 1.

$\gamma = 0$: Full privacy & $\gamma = 1/2$ is no privacy.



How useful is the new dataset, what is our estimate of the true mean?

$$\mathbb{E}[\overline{X_i}] = \left(\tfrac{1}{2}+\gamma\right)X_i + \left(\tfrac{1}{2}-\gamma\right)X_i = \left(\tfrac{1}{2}-\gamma\right) + 2\gamma \cdot X_i \Rightarrow X_i = \frac{\mathbb{E}[\overline{X_i}] - \left(\frac{1}{2}-\gamma\right)}{2\gamma}$$

$$\longrightarrow \frac{X_1+\ldots+X_n}{n} = \mathbb{E}\left[\frac{\frac{\overline{X_1}+\ldots+\overline{X_n}}{n} - \left(1/2-\gamma\right)}{2\gamma}\right]$$

↑ true mean ↑ estimate on the released data

$$\longrightarrow \Pr\left[\,|P-\overline{P}| > \varepsilon\,\right] \le \frac{\varepsilon}{4\cdot\gamma\cdot\sqrt{n}}$$ the probability that the difference is within ε.

Suppose we want ε-accuracy with at least 75% chance: we need $\frac{\varepsilon}{\gamma\sqrt{n}} \le 1$ or at least $n \ge \frac{1}{\varepsilon^2\cdot\gamma^2}$ people and we get decent individual level privacy.

---

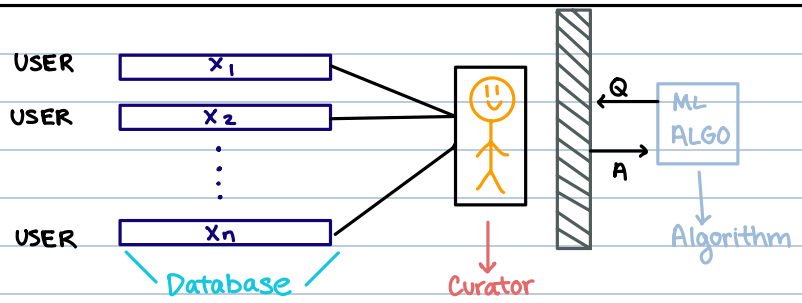
1) We must first quantify privacy
2) How will it effect the accuracy of our model?
   → For a fixed accuracy parameter how big should the dataset be to have the privacy guarantee.

Differential Privacy:

→ "Central Differential Privacy"

→ Trusted Curator model

The curator will then be asked a question in which it will be answered in a way that preserves privay.

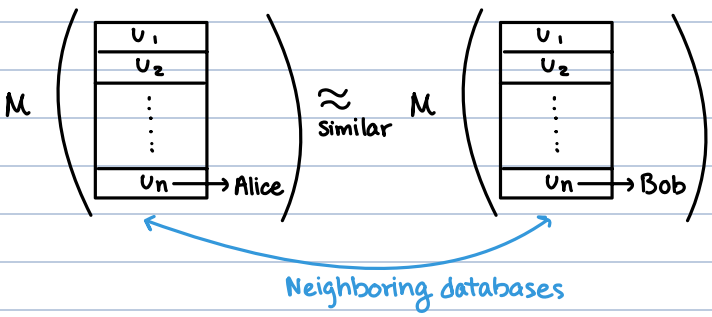


Designing a Curator:

→ Takes the database, and a query and provides an output for the query in a way that preserves privacy; goes from from: $X^n \longrightarrow Y : (\mathbb{R}^d)^n \longrightarrow \mathbb{R}$. The method of releasing a query is called a mechanism.

→ What is not private: If we cannot tell wether a user was part of the database.

Our answer for our mechanism should not change if we change a single user.

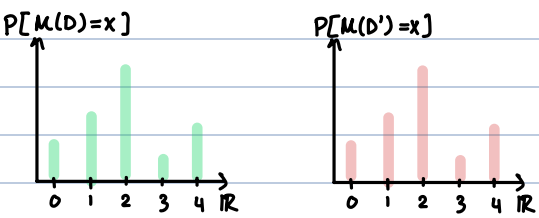


Neighboring Database: $D, D^1 \in X^n$ are neighbors if they differ only in one row

Similarity: Supposse there is a mechanism $M: X^n \to Y$ is ε-differentially private if $\forall$ neighboring databases D, D^1 $\forall y \in Y$: $e^{-\varepsilon} Pr[M(D')=y] \leq Pr[M(D)=y] \leq e^{\varepsilon} Pr[M(D')=y]$ we say that $e^{\varepsilon} \approx 1+\varepsilon$ for smaller ε

We also say smaller ε are more private.

→ The chance that the output is the same is epsilon-multiplicativity close



the heights are within e^{ε} of each other

Randomized Response:

Randomized Response is $O(\gamma)$ differentially private

Proof: for all possible $a \in \mathbb{R}$ we will have:

If $\overline{X_i} = \overline{X_i'}$ then the answers will match:

$$\frac{Pr[M(D)=a]}{Pr[M(D')=a]} \leq \frac{\frac{1}{2}+\gamma}{\frac{1}{2}-\gamma} = \frac{1+2\gamma}{1-2\gamma} = \left(1+\frac{4\gamma}{1-2\gamma}\right) \approx e^{\frac{4\gamma}{1-2\gamma}}$$

This is because since we are changing one entry we are only able to flip with rates $\frac{1}{2}+\gamma$ and $\frac{1}{2}-\gamma$

To have ε-dp and error $\leq \alpha$, randomized response needs a size of database $\geq \frac{1}{\varepsilon^2 \cdot \alpha^2}$

Perturbation:

Input Perturbation: We add noise to the input and then compute f(x_noise)

Output Perturbation: We compute f(x) and then add noise: f(x) + noise.

Laplacian noise:

Performs output perturbation, on the output.

Laplacian distribution: laplace with mean 0, and variance b is the distribution where $P(x) = \frac{1}{2b}\exp\left(\frac{-|x|}{b}\right)$ and the $\mathbb{E}[x] = \mu = 0$.

Claim: a laplacian mechanism with $b = \frac{1}{\varepsilon n}$ satisfies ε-differential privacy for releasing the mean:

Proof:

$M(x) = f(x) + z$ & $M(x') = f(x') + z$. Since x and x' differ by at-most one coordinate their difference in values will be at most : $\frac{\max(x' \text{ or } x)}{n} \Rightarrow |f(x) - f(x')| \leq \frac{1}{n}$. Now note that $M(x)$ will still be a laplacian distribution with mean of $f(x)$:

$$Pr[M(x) = a] = Pr[f(x) + z = a] = Pr[z = a - f(x)] = \frac{1}{2b} \cdot e^{-\frac{|a-f(x)|}{b}}$$

$$Pr[M(x') = a] = Pr[f(x') + z = a] = \frac{1}{2b} \cdot e^{-\frac{|a-f(x')|}{b}}$$

Then: $\frac{Pr[M(x)=a]}{Pr[M(x')=a]} = e^{\frac{-1}{b}(|a-f(x)| - |a-f(x')|)} \leq e^{1/bn} = e^{\varepsilon}$ where $b = \frac{1}{\varepsilon n}$

What is the accuracy of laplacian mechanism:

$Pr[|z| > t \cdot b] \leq \exp(-t)$

$Pr[|z| > 2 \cdot b] \leq \exp(-2) \leq \frac{1}{4}$

Hence: the leplacian mechanism with $b = \frac{1}{\varepsilon n}$ is ε-differentially private and $Pr[M(x) - f(x) > \frac{2}{\varepsilon n}] \leq \frac{1}{4}$

↳ To get ε-DP and accuracy α, $n \geq \frac{2}{\varepsilon \cdot \alpha}$

Sensitivity: $f: X^n \to \mathbb{R}$ then $S_1(f) = \max_{x, x'} |f(x) - f(x')|$. for univariate output mean: $f(\text{Mean}) = \frac{1}{n}$

$f: X^n \to \mathbb{R}^n$ then $S_1(f) = \max_{x\ x'} \sum_{i=1}^{k} |f(x)_i - f(x')_i|$

Thm: If $f: X^n \to \mathbb{R}$ that has $S_1(f) \leq 8$. Then laplacian mechanism with noise $b = 8/\varepsilon$ achieves ε-DP this also holds for multiple outputs.

Generalized Laplacian Mechanism:

⟶ Compute $f(x)$

⟶ Compute k-independent laplacian noise variables $z_1, \ldots, z_k \sim$ laplacian(b)

⟶ Output $f(x) + (z_1, z_2, \ldots, z_k)$

If $f: X^n \to \mathbb{R}^k$ has sensitivity $S_1(f) \leq 8$ then laplacian mechanism with noise rate, $b = 8/\varepsilon$ achieves ε-DP

Remarks:

The higher the sensitivity the more difficult it becomes to preserve privacy.

Post Processing and Privacy:

→ After answering a query we must not care what happens after, and our answer must still be private.

Thm: Let $M: X^n \to Y$ is ε-differentially private then, let $F: Y \to Z$. Then $F \circ M: X^n \to Z$ is also ε differentially private.

Proof:

for every $z \in Z$: $\Pr[F \circ M(x) = z] = \sum_{y: F(y)=z} \Pr[M(x)=y] \leq \sum_{y: F(y)=z} e^{\varepsilon} \cdot \Pr[M(x')=y]$ from M being ε-differentially private

$$\Rightarrow e^{\varepsilon} \cdot \sum_{y: F(y)=z} \Pr[M(x')=y] = e^{\varepsilon} \Pr[F \circ M(x') = z]$$

Group Privacy: suppose we have two databeses X, X' that differs in at most k-rows: Say there is a mechanism $M: X^n \to V$ is ε-differentially private $\forall y$. then:

$$\Pr[M(x)=y] \leq e^{k \cdot \varepsilon} \Pr[M(x')=y]$$

Proof:

We use intermediary databases where we change only one neighbor between each database: let $x^0 \dots x^k$ denote the intermediary databases:

$$\Pr[M(x^0)=y] \leq e^{\varepsilon} \cdot \Pr[M(x')=y]$$
$$\Pr[M(x')=y] \leq e^{\varepsilon} \cdot \Pr[M(x^2)=y]$$
$$\vdots$$
$$\Pr[M(x^{k-1})=y] \leq e^{\varepsilon} \cdot \Pr[M(x^k)=y] \Rightarrow \Pr[M(x')=y] \leq e^{k \cdot \varepsilon} \Pr[M(x^k)=y]$$ by telescoping series.

Counting Queries:

Reveal # of users or records in the database that have certain properties: the sensitivity will be at most 1. For any counting query we can get ε-differential privacy by adding $\text{lap}(\frac{1}{\varepsilon})$ noise.

k-Counting Queries:

If we allow k-counting queries then our sensitivity is k. Hence if we add $\text{lap}(\frac{k}{\varepsilon})$ noise gives us ε-differential privacy

---

Histogram Queries:

We want to produce histogram queries if each user is within one of k-groups. The sensitivity of a histogram is: $S_1(\text{HQuery}) \leq 2$ since we can remove a person and add that person to another group. $\Rightarrow$ Adding $\text{lap}(\frac{2}{\varepsilon})$ gives us ε-differential privacy.

Accurracy Analysis:

Say $z = \text{lap}(b)$. Then $\Pr[|z| > b \cdot t] \leq e^{-t}$ ⟶ we can use this to determine how this affects our accurracy.

Supposse $M_1, \dots, M_k$ are ε-dP then their composition $M_1 \circ M_2 \circ \dots \circ M_k$ will be $k \cdot \varepsilon$-DP

Proof: $\Pr[M_k(x)=y] \leq e^{\varepsilon} \cdot \Pr[M_k(x')=y]$

$$\Pr[M_{k-1}(M_k(x))=z] \leq e^{\varepsilon} \cdot e^{\varepsilon} \cdot \Pr[M_{k-1}(M_k(x'))=z]$$
$$\vdots$$

Noisy Max:

given a $f: X^n \to \mathbb{R}^k$ where $g(x) = \arg\max_i f(x)_i$. We are trying to output the maximum. Noisy max, adds noise $\text{lap}(\frac{1}{\varepsilon})$ to each coordinate, we compute the max of these noisy counts. Then outputs the id of the object with the highest noisy max.

Thm: Noisy max is ε differentially private

Exponential Mechanism

→ We have a range P

→ We have a utility function $U: X^n \times P \to P$: $U(x,p)$: "utility at price p" our goal is compute and release $\arg\max_{p \in P} U(x,p)$

Ex: Digital goods auction

Maximum, we can define a utility function that counts the number of occurrances.

Given a utility function $U: X^n \times P$, $M_\varepsilon(x)$ selects and outputs $p \in P$ with probability proportional to:

$$\exp\left(\frac{\varepsilon \cdot U(x,p)}{\Delta u}\right) \quad \text{where} \quad \Delta U = \max_p \left[\max_{x,x'} |U(x,p) - U(x',p)|\right]$$

Claim: We have that M_E is (2ε)-Differentially private

Ex: Computing most common form of given cancer:

$U: X \times P \to \mathbb{R}$ and $U(x, id) = \#$ of people with type id, then $\Delta U \leq 1$, now we would output i with probability proportional to $\exp(\varepsilon \cdot u(x,i))$

→ Output i with probability $\frac{\exp(\varepsilon \cdot u(x,i))}{\sum_{j=1}^{n} \exp(\varepsilon \cdot u(x,j))}$

Proof that M_E is differentially private:

$\Pr[M_E(x) = p]$ compared to $\Pr[M_E(x') = p]$

$$\Pr[M_E(x) = p] = \frac{\exp\left(\frac{\varepsilon \cdot u(x,p)}{\Delta U}\right)}{\sum \exp\left(\frac{\varepsilon \cdot u(x,q)}{\Delta U}\right)} \quad \& \quad \Pr[M_E(x) = p] = \frac{\exp\left(\frac{\varepsilon \cdot u(x',p)}{\Delta U}\right)}{\sum \exp\left(\frac{\varepsilon \cdot u(x',q)}{\Delta U}\right)}$$

Recall by definition: $\frac{|u(x,q) - u(x',q)|}{\Delta u} \leq 1$

Comparing the numerators:

$$\frac{\exp\left(\frac{\varepsilon \cdot u(x,p)}{\Delta u}\right)}{\exp\left(\frac{\varepsilon \cdot u(x',p)}{\Delta u}\right)} \leq \exp\left(\frac{\varepsilon \cdot |u(x,p) - u(x',p)|}{\Delta u}\right) \leq \exp(\varepsilon) = e^\varepsilon$$

Comparing denominators:

$$\sum \exp\left(\frac{\varepsilon \cdot u(x,q)}{\Delta u}\right) \geq \sum \exp\left(\frac{\varepsilon \cdot u(x',q)}{\Delta u}\right) \cdot e^{-\varepsilon}$$

$$\Rightarrow \sum \exp\left(\frac{\varepsilon \cdot u(x,a)}{\Delta U}\right) \leq e^\varepsilon \sum \exp\left(\frac{\varepsilon \cdot u(x,a)}{\Delta u}\right)$$

$$\Rightarrow \frac{\Pr[M_E(x) = p]}{\Pr[M_E(x') = p]} = \frac{e^\varepsilon}{e^{-\varepsilon}} = e^{2\varepsilon}$$

Accurracy of Exponential Mechanism:

$$Pr\left[\underbrace{u(x, M_E(x))}_{\text{utility achieved by exponential mech}} \leq OPT_u(x) - \frac{\Delta u}{\epsilon}(\ln|P| + t)\right] \leq e^{-t}$$

optimal utility.

Says that the probability we are some γ distance away from the optimal utility is less than e^{-t}

Approximate differential Privacy

Pure differential privacy can be hard to achieve, if we have very small probabilities

A mechanism $M: X^n \to Y$ is (ϵ, δ)-Differentially private if $\forall y \in Y$ and neighboring databases x, x' we have:

$Pr[M(x)=y] \leq e^{\epsilon} \cdot Pr[M(x')=y] + \delta$

In essence we have an additive error, but δ must be very small. We have privacy except with probability delta

Ex: M: with probability $1-\delta$ output junk
with probability δ output entire database

this is $(0, 2\delta)$-DP since $Pr[M(x)=x] = \delta$
$Pr[M(x')=x] = 0$ since the databases are different

Ex: With probability $(1-\delta)$ we replace their record with junk, and with probability δ we keep their record, this will end up having $(0, 2\delta)$ privacy. If $\delta << 1/n$ the number of records that actually survive is δn

L_2-Sensitivity:

$f: X^n \to \mathbb{R}^k$ we define $S_2(f) = \max_{x,x'} \|f(x) - f(x')\|_2 = \max_{x,x'} \left(\sum_{i=1}^{k} (f(x)_i - f(x')_i)^2\right)^{1/2}$

Ex: we have samples $x_i \sim \{0,1\}^d$ and $f(x) = \frac{1}{n}\sum_{i=1}^{n} x_i$ (the average) then $S_1(f) = \|f(x) - f(x')\| = \frac{d}{n}$ but observe $S_2(f) = \sqrt{d}/n$

$$\max_{x,x'} \|f(x) - f(x')\|_2 = \left\| \frac{1}{n}(x_1 + x_2 + \ldots + x_n) - \frac{1}{n}(x_1 + \ldots + \bar{x}_n) \right\|_2 = \left\| \frac{1}{n}(x_n - \bar{x}_n) \right\|_2 = \sqrt{d}/n$$

Guassian Mechanism:

→ $f: X^n \to \mathbb{R}^k$

→ sample $(z_1, z_2, \ldots, z_k)$ that are independent and identically distributed $N(0, \sigma^2)$

→ Output $f(x) + (z_1, \ldots, z_k)$

Thm: guassian mechanism with $\sigma \geq \sqrt{2 \cdot \ln\left(\frac{1.25}{\delta}\right)} \cdot \frac{S_2(f)}{\epsilon}$ will be (ϵ, δ) differentially private. This follows from working and comparing the pdf of guassian distributions.

Ex: for the above function for ϵ-DP we add noise $\approx \frac{d}{n \cdot \epsilon}$ compared with a guassian distribution $\approx \sqrt{\ln\left(\frac{1}{\delta}\right)} \cdot \frac{\sqrt{d}}{n \cdot \epsilon}$.

Hence for accurracy we get almost a quadratic improvement in the number of samples needed.

If x and x' differ in k-rows then M is (ϵ, δ)-DP if $Pr[M(x)=y] \leq e^{k \cdot \epsilon} Pr[M(x')=y] + k \cdot e^{k\epsilon} \delta$

Basic Composition thm:

If $M_1, M_2, \ldots, M_k$ are (ϵ, δ)-DP then their composition is $(k \cdot \epsilon, k\delta)$-DP

Advanced Composition thm:

If $M_1, \ldots, M_k$ are ϵ-DP then their composition is $(\sqrt{2k \ln(1/\delta)} \cdot \epsilon + k\epsilon^2, \delta)$-DP

The advanced composition thm allows for k^2 more adaptive queries

Private Machine Learning:

How do we mitigate the issues of ML models memorizing information and then leaking information?

Recall that ML is empirical risk minimization with some parametric family h_θ, our goal is to make ERM private.

→ Output perturbation: compute θ, release θ + noise

→ Input perturbation: perturb the input

→ Objective perturbation: add noise to the loss function

→ Gradient perturbation: add noise to each gradient computation.

Differentially Private SGD:

DP-SGD:

Pick a start θ_0

For $t = 1, \ldots, T$:

Pick index $i \in [n]$ at random

$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \ell(h_\theta(x_i), y_i) + z_T$ (Guassian Noise)

Privacy still requires a centralized trusted center party:

Local differential privacy: we want $\forall v, v'$ distinct $Pr[\text{Message}(v) = z] \leq e^\epsilon \cdot Pr[\text{Message}(v) = z]$ this means that we want privacy for each user so that a central server cannot tell what exactly each user has. Randomized response satisfies ldp.

Other forms include:

→ label differential privacy

→ User-level differential privacy

→ shuffle differential privacy

Graphical Models:

This is an unsupervised learning method: given a dataset, we want to build a "model" for the dataset

→ Probabilistic models of data, these are also precursors of deep generative models, the goal of these models is to determine dependencies, casual structure in the data

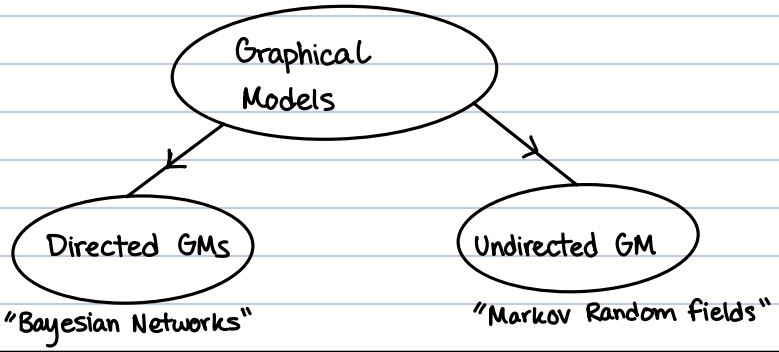
Independence: $X \perp Y \Leftrightarrow Pr[X=x, Y=y] = Pr[X=x] \cdot Pr[Y=y] \Leftrightarrow Pr[X=x|Y=y] = Pr[X=x]$

Independence is not perfect for determining a relation since we deal with hidden variables, instead we want to look at conditional independence.

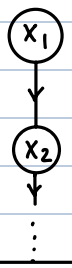
Conditional Independence $X \perp Y \mid Z \Leftrightarrow Pr[X=x, Y=y \mid Z=\delta] = Pr[X=x \mid Z=\delta] Pr[Y=y \mid Z=\delta]$

$\Leftrightarrow Pr[X=x \mid Y=y, Z=\delta] = Pr[X=x \mid Z=\delta]$

Graphical Models are distributions with "constrained" conditional independence relations



Markov Chain:



then $X_3 \perp X_1 \mid X_2$

$X_4 \perp X_2 \mid X_3$

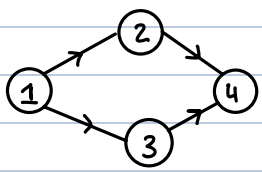
$X_4 \perp X_1 \mid X_3$

Bayesian Network

→ A directed acyclic graph on [d] verticies in G. A distribution D is a bayesian network with graph G if:

$$Pr[X=(x_1,\dots,x_d)] = \prod_{i=1}^{d} Pr[x_i = x_i \mid \text{Parents of } i \text{ in } G]$$

Ex:

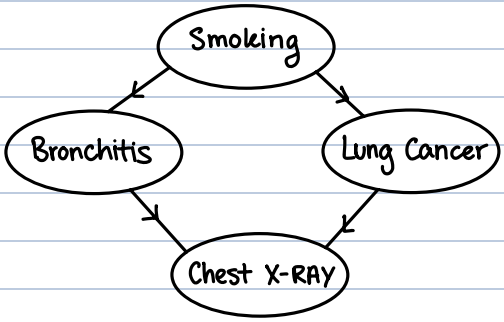


then $Pr[X=(x_1,x_2,x_3,x_4)] = Pr[X_1=x_1] \cdot Pr[X_2=x_2 \mid X_1=x_1] \cdot Pr[X_3=x_3 \mid X_1=x_1] \cdot Pr[X_4 \mid X_3, X_2]$

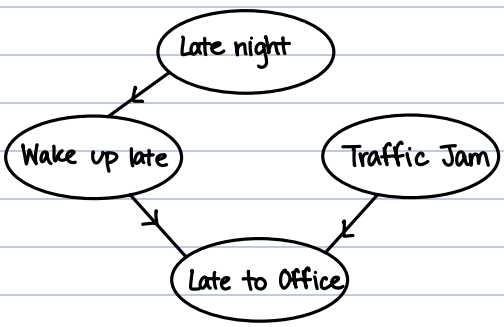
Ex: Markov Chain:

then $Pr[X=(x_1,x_2,\dots x_d)] = Pr[X_1=x_1] Pr[X_2=x_2 \mid X_1=x_1] Pr[X_3=x_3 \mid X_2=x_2] Pr[X_d \mid X_{d-1}]$

Ex: Baysian Networks help model dependencies (Dependency Graphs)



$Pr[X=(S,B,L,C)] = Pr[S] Pr[L|S] Pr[B|S] Pr[C|B,L]$



$Pr[X=(L,W,O,T)] = Pr[L] Pr[T] Pr[W|L] Pr[O|W,T]$

Learning Goal

→ Input: Given n-samples $x^1, x^2, \ldots, x^n \in \sum^d$

→ Output: the underlying directed dependency graph & the right conditional distributions.

(Refer to appendix d for bayes nets)

The learning problem of just creating a bayes network is not structered enough. Instead we would want to impose **structural limitations** on our output to make it more meaningful. eg (in-degree is bounded, the graph has to be a tree, path, grid, ..., etc.)

This is because we could simply create a fully connected distribution and return that however we won't learn anything about the actual dependencies, meaning we can connect every node to every other node.

Chow-Liu-Algorithm:

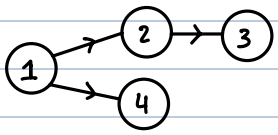
→ We can learn tree-shaped bayesian networks. Our input is $x^1, x^2, \ldots, x^n \in \Sigma^d$ which are samples from some distribution d.

→ **Assumption:** D has a bayesian network which is a "rooted tree"

A rooted tree: every single node except the parent has a single parent

→ **Goals:** we output the true tree, output the right conditional distribution

structure learning ↑ parameter learning ↑



or

Markov Chains are rooted trees

We want to learn a distribution D' such that dist(D, D') is small

KL-Divergence: measures how close two distributions are (cross-entropy)

$$KL(P\|Q) = \sum_a P(a) \cdot \log_2 \frac{P(a)}{Q(a)} = \sum_a P(a) \log_2 P(a) - \sum_a P(a) \cdot \log_2 Q(a)$$

probability that a happens under P

probability that a happens under Q

Entropy of a distribution

Example: $\Omega = \{0, 1\}$ and $p(0) = 1/2$; $q(0) = 1/3$; $p(1) = 1/2$; $q(1) = 2/3$

$$KL(p\|q) = P(0) \cdot \log_2 \frac{P(0)}{q(0)} + P(1) \log_2 \frac{P(1)}{q(1)} = \frac{1}{2}\log(3/2) + \frac{1}{2}\log(3/4) = \frac{1}{2}\log(9/8)$$

We also note that $KL(P\|Q) \geq 0$ and $KL(P\|Q)$ is zero if and only if $p = q$

Reformulating Chow-Liu:

Input: Samples from a distribution P^* on $\sum^d$ generated by a Bayes net on a tree T^*.

Output: Find a tree T and a corresponding Bayes net P_T s.t. $KL(P^* \| P_T) \leq \varepsilon$

We first start with an intermediary problem where we are given samples P^* and the correct Bayes net T come up with the right conditional distributions.

Solving Baby Chow-Liu:

Given the tree T we can use the samples to estimate $P(x_i|x_j)$ and we use that

$\longrightarrow P(x_i=a \mid x_j=b) = \dfrac{P(x_i=a \wedge x_j=b)}{P(x_j=b)}$

$\longrightarrow$ Over *all* samples count how many times $x_i=a$ and $x_j=b$

$\longrightarrow$ Over *all* samples count how many times $x_j=b$

$\longrightarrow$ And we generate $\hat{P}(x_i|x_j)$

$\longrightarrow$ The estimator for the entire tree: $\hat{P}_T = \prod_{e=(u,v)} \hat{P}(x_v|x_u)$

There are two possible sources of error: the estimated value v.s. working with the right tree.

$\hookrightarrow$ This is easy to control with $1/\epsilon^2$ samples our conditional probability estimates will have error $\leq \epsilon$

Hence we must try and figure out the right tree to minimize K.L. divergence

$$KL(P_{T^*}\|P_T) = (\text{Entropy of } T^*) - \sum_a P_{T^*}(a)\log P_T(a) = (\text{Entropy of } T^*) - \sum_a P_{T^*}(a) \sum_{(u,v)\in T} P(x_v=a|x_u=a)$$

$$= (\text{Entropy of } T^*) - \sum_{(u,v)\in T} \left(\sum_a P_{T^*}(a)\cdot P(x_v=a_v|x_u=a_u) \right)$$

$$= J_* - \sum_{(u,v)\in T} I(x_u;x_v)$$

where J_* only depends on only T^*

$I(X;Y) \equiv$ Mutual information between X and Y

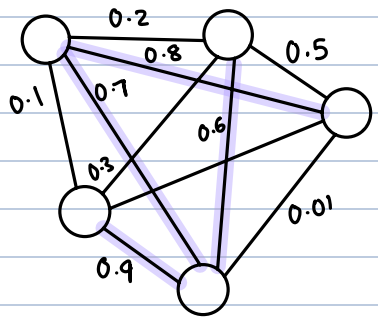
$$I(X;Y) = \sum_{a,b} Pr[X=a, Y=b]\cdot \log \frac{Pr[X=a, Y=b]}{Pr[X=a]Pr[Y=b]}$$

Aside on $I(X,Y)$ if X,Y are completely dependent on each other e.g. $I(X,2X) \equiv$ "Full" denoted $H(X)$. If X and Y are independent then $I(X;Y)=0$ therefore $H(X) \geq I(X;Y) \geq 0$.

Thm: Given any tree T $KL(P_{T^*}\|P_T) = J_{P^*} - \sum I(x_u;x_v)$. Since our original goal was find T to minimize $KL(P_{T^*}\|P_T)$ this will be equivalent to maximizing $\sum_{(u,v)\in T} I(x_u;x_v)$. We don't know $I(X,Y)$ exactly but we can estimate with ϵ given $1/\epsilon^2$ samples

Given such a graph with edge waits are $I(x_i;x_j)$ we want to maximize this:

$\longrightarrow$ Solving maximum spanning tree



Grown-Up Chow-Liu-Algorithm:

- Use the samples to estimate $I(x_i,x_j)$ for every (i,j) upto some error ϵ
- Form a weighted graph G where weights are exactly the estimated mutual information values
- Compute T_{CL} to be the maximum spanning tree in G
- Output $P_{T_{CL}}$ as your approximating distribution

Thm: Given a graph on d-verticies we can find a max spanning tree in $O(d^2)$

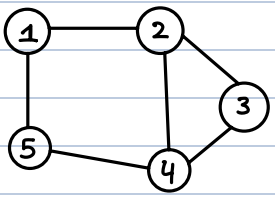
Thm: If we have $O(d^2/\epsilon^2)$ samples, then CHOW-LIU algorithm gives a tree $KL(P_{T^*}|P_T) \leq \epsilon$ and run-time $O(d^2)$

Undirected Graphical Models:

→ Called Markov Random Fields" (MRFs)

We have a distribution $D=(X_1, X_2, \dots, X_d)$ and a dependency graph G for D. Then D satisfies Pairwise Markov Properties with respect to G if (i,j) that have no edge $X_i \perp X_j \mid X_{rest}$. We say that D satisfies Local Markov Properties with respect to G if (i,j) not an endge then $X_i \perp X_j \mid X_{neighbors\ of\ i}$. We say that D satisfies Global Markov Properties with respect to G if (i,j) not an endge then $X_i \perp X_j \mid$ Seperating Set. A seperating is a Subset of any verticies s.t. removing them disconnects i,j

Ex:



$X_1 \perp X_4 \mid X_2, X_3, X_5$
$X_1 \perp X_3 \mid X_2, X_4, X_5$
$X_2 \perp X_5 \mid X_1, X_3, X_5$
↑ Pairwise

$X_1 \perp X_4 \mid X_2, X_5$
$X_1 \perp X_3 \mid X_2, X_5$
$X_2 \perp X_5 \mid X_1, X_4$
↑ local

$X_1 \perp X_3 \mid X_2, X_4$
$X_1 \perp X_4 \mid X_2, X_5$
↑ Global

We note that the distribution Global MP ⇒ local MP ⇒ Pairwise MP. Hence for most distributions we will have all 3.

Take a markov chain: then PMP: $X_1 \perp X_3 \mid X_2, X_4, \dots, X_d$; LMP: $X_1 \perp X_4 \mid X_3, X_5$; GMP: $X_2 \perp X_4 \mid X_3$

The main challenges for learning:

→ Structure learning: learn graph G

→ Parameter learning: learn the distribution G

→ Inference: output $X_i \mid X_{partial\ assignement}$

Structure Learning:

The problem would be ill-posed in the case where we have no restrictions since a fully connected graph would be vacously true.

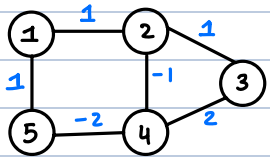
We make an assumption that our graph is sparse! since if we have missing edges it then gives us a reason to model it's dependencies:

Input: $X^1, X^2, \dots, X^n \sim D$

Output: Dependency graph of D where each vertex has degree $\leq k$

Boltzman Machines:

Can be thought as discrete models we have $D \approx (X_1, \dots, X_d)$ where $X = \{1, -1\}$ then $Pr[X=(x_1,\dots,x_d)] \propto \exp\left(\sum_{\{i,j\} \in G} w_{ij} x_i \cdot x_j\right) \propto \prod_{\{i,j\} \in G} \exp(w_{ij} x_i \cdot x_j)$ we are saying the probability is based on the weights of the graph G.



$Pr[X=(x_1,x_2,\dots,x_5)] \propto \exp(1 \cdot x_1x_2 + 1 \cdot x_2 \cdot x_3 - x_2 \cdot x_4 + 2x_3x_5 + x_1x_5 - 2x_4x_5)$

Given a weighted graph we can define a distribution for that graph and vice-versa

Guassian Graphical Models:

Real-valued distribution: D on $\mathbb{R}^d \rightsquigarrow N(0, \Sigma)$ where $\Sigma_{ij} = \mathbb{E}[X_i \cdot X_j]$ and Σ is the covariance matrix.

The precision matrix: $\Theta = \Sigma^{-1}$ (inverse of covariance matrix)

Thm: The support of Θ gives a dependency graph of $N(0, \Sigma)$. The support is defined as $\{i,j\} \in G$ if $\Theta_{ij} \neq 0$. If the entry is non-zero then we have to add an edge to the graph G.

For a guassian distribution: $Pr[X = x] \propto \exp(-x^T \Sigma^{-1} x)$