Warm up

$j = 1, \dots, p$

$x_i \to h_j(x_i) = \cos(g_j^T x_i + b_j)$

$X = \begin{bmatrix} - x_1 - \\ \vdots \\ - x_n - \end{bmatrix} \in \mathbb{R}^{n \times d}$

$H = \begin{bmatrix} - h_1 - \\ \vdots \\ - h_n - \end{bmatrix} \in \mathbb{R}^{n \times p}$

1 float in NumPy = 8 bytes

$10^6 \approx 2^{20}$ bytes = 1 MB

$10^9 \approx 2^{30}$ bytes = 1 GB

```
# generate some nonsense data for an example
X = np.random.randn(n,d)
y = np.random.randn(n)
```

$n \times d$

$H^T H = \sum_{i=1}^{n} h_i h_i^T$

```
# generate the random features
G = np.random.randn(p, d)*np.sqrt(.1)
b = np.random.rand(p)*2*np.pi
```

```
# construct HTH
HTH = np.zeros((p,p))
HTy = np.zeros(p)
for i in range(n):
    hi = cos(np.dot(X[i,:], G.T)+b)
    HTH += np.outer(hi, hi)
    HTy += y[i]*hi
    if i % 1000==0: print(i)
```

$p \times p$

$nd + p^2$

```
H = cos(np.dot(X, G.T) + b.T)
HTH = np.dot(H.T, H)
HTy = np.dot(H.T, y)
```

$n \times p$

$p \times p$

$nd + np + p^2$

```
# construct HTH
HTH = np.zeros((p,p))
HTy = np.zeros(p)
block = p
for i in range(int(np.ceil(n/block))+1):
    Hi = cos(np.dot(X[i*block:min(n,(i+1)*block),:], G.T)+b)
    HTH += np.dot(Hi.T, Hi)
    HTy += np.dot(Hi.T, y[i*block:min(n,(i+1)*block)])
```

```
w = np.linalg.solve(HTH + lam*np.eye(p), HTy)
```

For each block compute the memory required in terms of n, p, d.

If d << p << n, what is the most memory efficient program (blue, green, red)?

If you have unlimited memory, what do you think is the fastest program?

Convexity

What is a convex set?

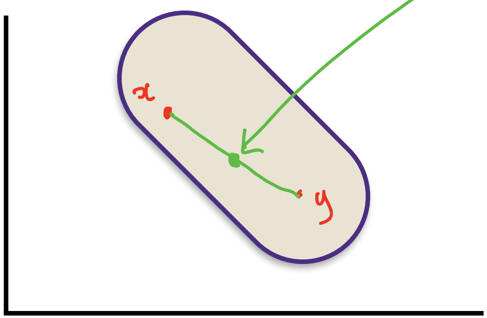
$K : \{ (x_1, x_2) : x_1^2 + x_2^2 \leq 10 \}$

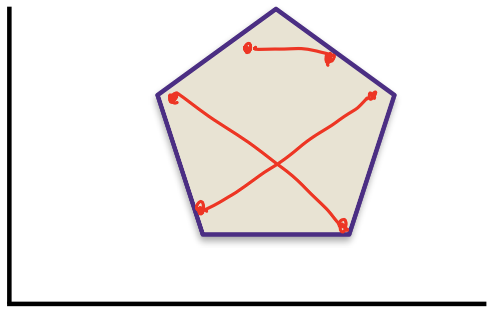
$(x_1, x_2), (y_1, y_2) \in K$

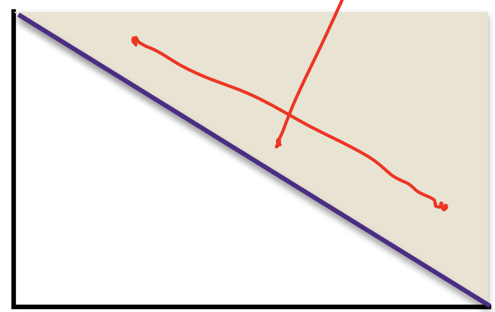
A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0, 1]$

Show $((1-\lambda)x_1 + \lambda y_1, (1-\lambda)x_2 + \lambda y_2)$
$\in K$

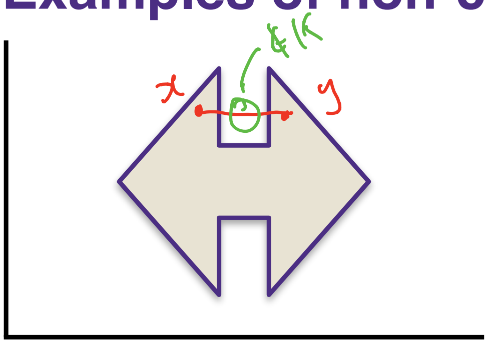
Examples of convex sets

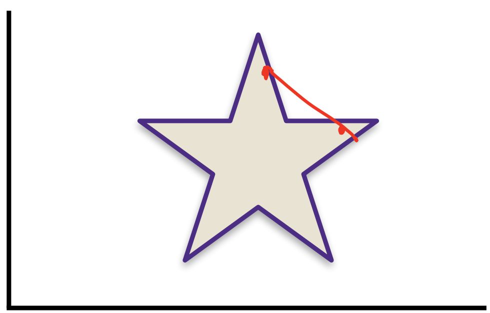


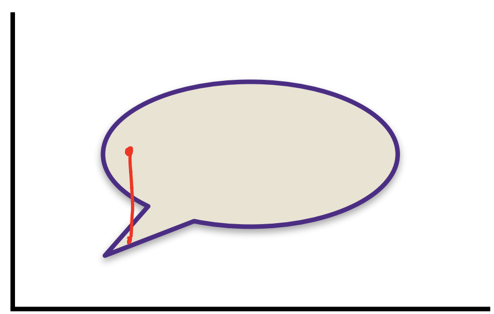




Examples of non-convex functions: anything else





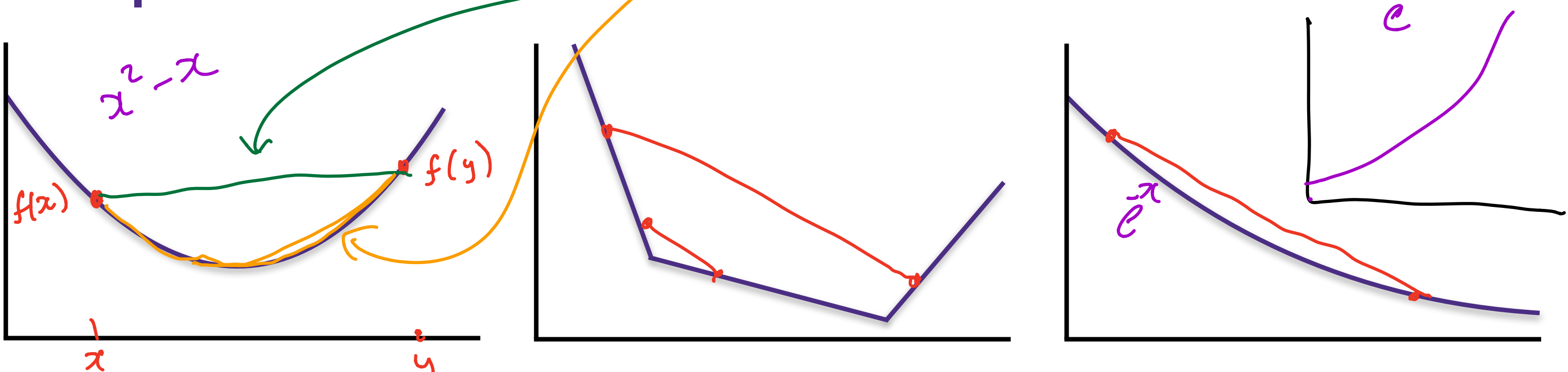


What is a convex function?

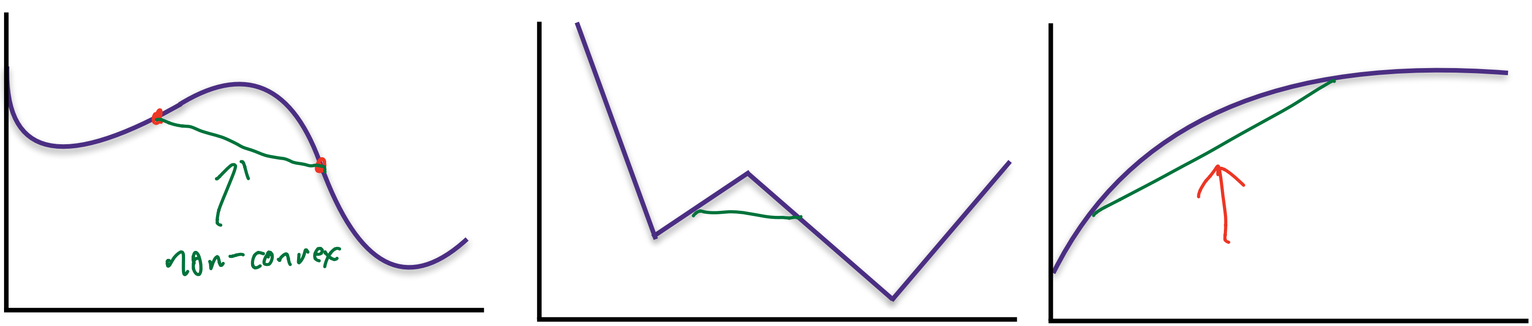
dom(f): {x : f(x) defined}

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if $f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y)$ for all $x, y \in \mathbb{R}^d$ dom(f) and $\lambda \in [0,1]$

Examples of convex functions: "look like bowls"



Examples of non-convex functions: anything else

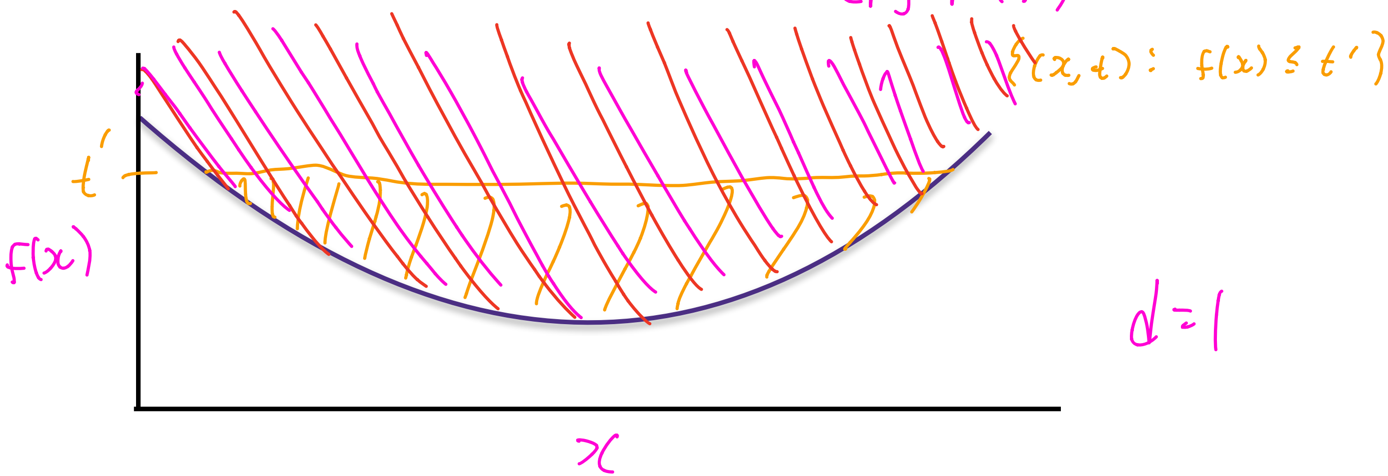


Convex functions and convex sets?

A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0,1]$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if $f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y)$ for all $x, y \in K$ and $\lambda \in [0,1]$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if the set $\underbrace{\{(x,t) \in \mathbb{R}^{d+1} : f(x) \leq t\}}_{\text{epigraph}(f)}$ is convex



More definitions of convexity

A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0,1]$

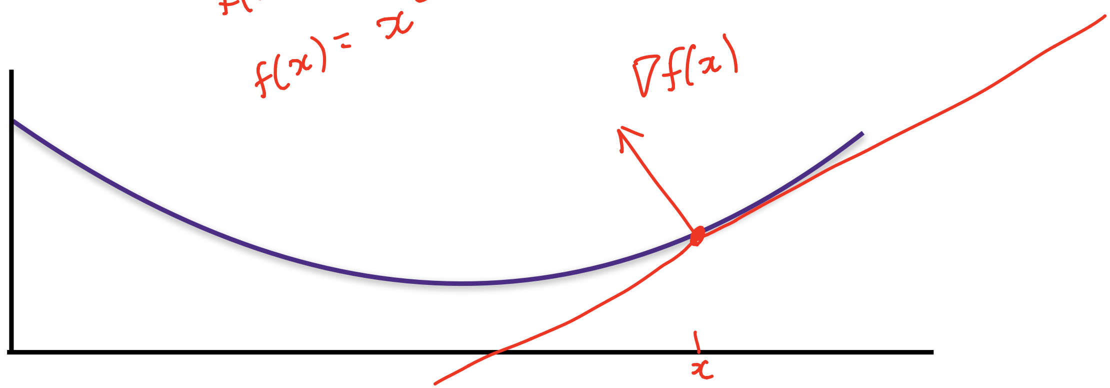
A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if the set $\{(x,t) \in \mathbb{R}^{d+1} : f(x) \leq t\}$ is convex

A function $f : \mathbb{R}^d \to \mathbb{R}$ that is differentiable everywhere is convex if $f(y) \geq f(x) + \nabla f(x)^\top (y - x)$ for all $x, y \in dom(f)$

Try at home

$f(x) = e^x$ show convex using ↑

$f(x) = x^2$



More definitions of convexity

$f(x) = \frac{1}{2}x^2$

$f'(x) = x$

$f''(x) = 1 > 0 \Rightarrow f$ is convex

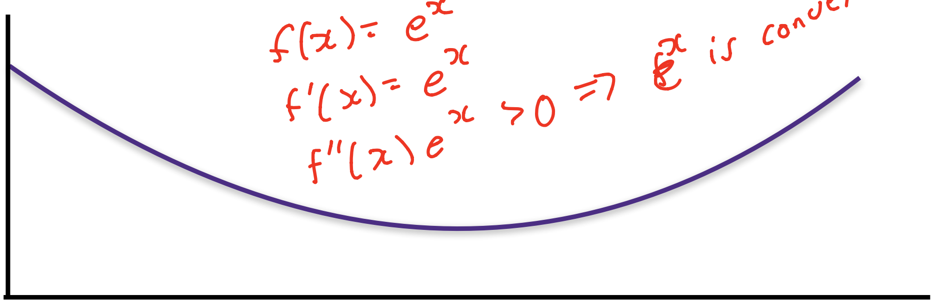
A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0,1]$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if the set $\{(x,t) \in \mathbb{R}^{d+1} : f(x) \leq t\}$ is convex

A function $f : \mathbb{R}^d \to \mathbb{R}$ that is differentiable everywhere is convex if $f(y) \geq f(x) + \nabla f(x)^\top (y - x)$ for all $x, y \in dom(f)$

A function $f : \mathbb{R}^d \to \mathbb{R}$ that is twice-differentiable everywhere is convex if $\nabla^2 f(x) \succeq 0$ for all $x \in dom(f)$

$[\nabla^2 f(x)]_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \, \partial x_j}$



$f(x) = e^x$

$f'(x) = e^x$

$f''(x) e^x > 0 \Rightarrow e^x$ is convex

$A \succeq 0$ if

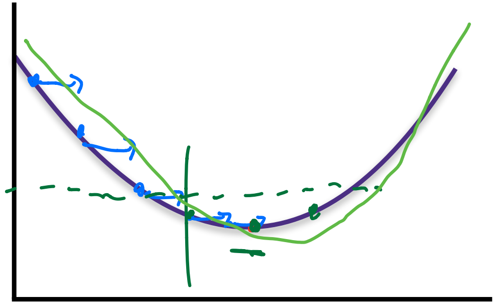
$x^\top A x \geq 0 \quad \forall x$

Why do we care about convexity?

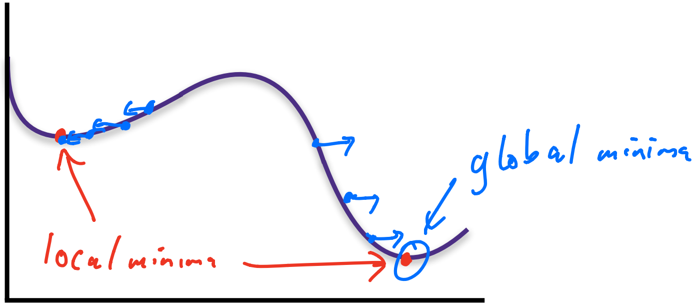
Convex functions

- All local minima are global minima
- Efficient to optimize (e.g., gradient descent)

Convex Function



Non-convex Function



Gradient Descent

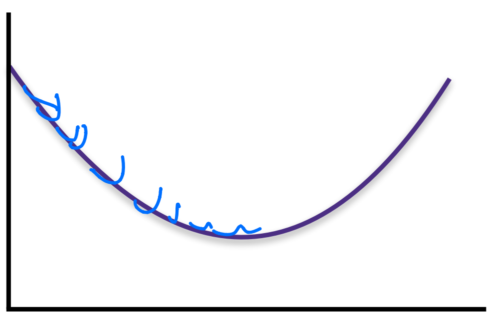
Initialize: $w_0 = 0$ (or randomly)

for $t = 1, 2, \ldots$

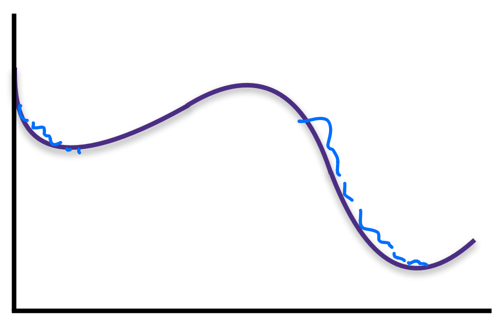
$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

step size

Convex Function



Non-convex Function



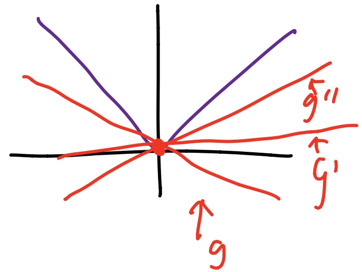
Sub-Gradient Descent

Initialize: $w_0 = 0$

for $t = 1, 2, \ldots$

Find any g_t such that $f(y) \geq f(w_t) + g_t^\top (y - w_t)$

$w_{t+1} = w_t - \eta g_t$



$f(x) = |x|$

f is not diff. at $x = 0$

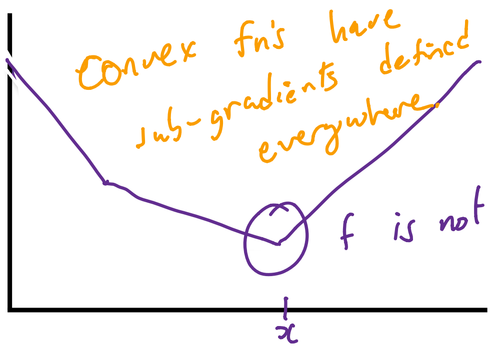
but $g \in [-1, 1]$ is a sub-gradient for $|x|$ at 0 since

$|y| \geq |x| + g(y - x)$

sub-gradients are not unique

g is a subgradient at x if $f(y) \geq f(x) + g^T (y - x)$

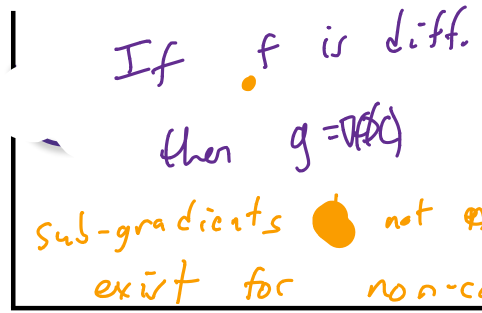
Convex Function



Convex fn's have sub-gradients defined everywhere

f is not differentiable

Non-convex Function



If f is diff. at x then $g = \nabla f(x)$

Sub-gradients not necessarily exist for non-convex fns

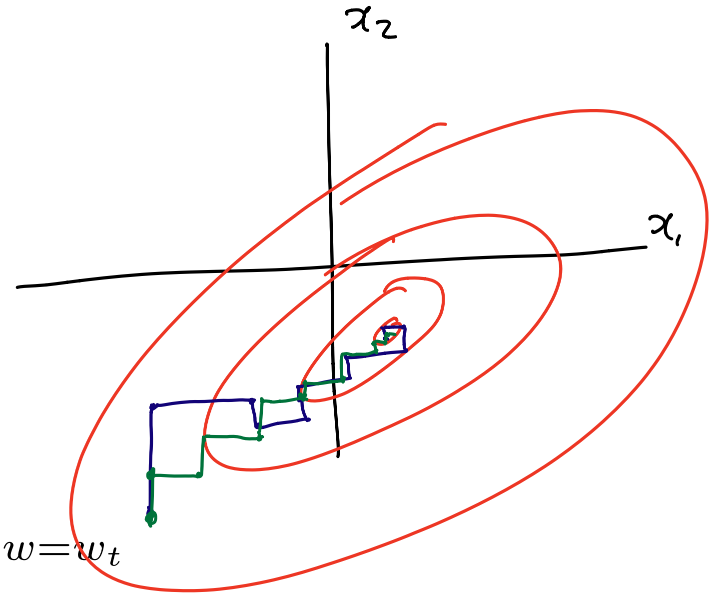
Coordinate descent

Initialize: $w_0 = 0$

for $t = 1, 2, \ldots$

Let $i_t = (t \,\%\, d) + 1$

$$w_{t+1}^{(i_t)} = w_t^{(i_t)} - \eta_t \frac{\partial f(w)}{\partial w^{(i_t)}}\bigg|_{w=w_t}$$



Special case:

Choose i_t : $\alpha_t = \arg\min_{\alpha} f(w_t + e_{i_t}\alpha)$

$w_{t+1}^{(i_t)} = w_t + e_{i_t}\alpha$

Machine Learning Problems

- **Given data:**

$$\underbrace{\{(x_i, y_i)\}_{i=1}^n}$$ $$x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- **Learning a model's parameters:** $$\sum_{i=1}^n \underbrace{\ell_i(w)}$$

each

If $\ell_i(w)$ is convex the sum is convex

$$\text{Logistic Loss: } \ell_i(w) = \underbrace{\log(1 + \exp(-y_i\, x_i^T w))}$$

$$\text{Squared error Loss: } \ell_i(w) = \underbrace{(y_i - x_i^T w)^2}$$

Gradient Descent:

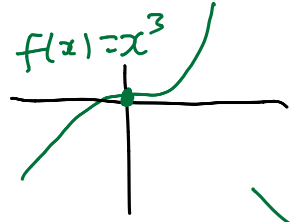
iteration t

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Bigg|_{w=w_t}$$

$$= w_t - \eta \frac{1}{n} \sum_{i=1}^n \nabla_w \ell_i(w) \Big|_{w=w_t}$$

Optimization summary

$f(x) = x^3$



$f'(x) = 0$ at $x = 0$

- You can always run gradient descent whether f is convex or not. But you only have guarantees if f is convex
- Many bells and whistles can be added onto gradient descent such as momentum and dimension-specific step-sizes (Nesterov, Adagrad, ADAM, etc.)

CVX - software package for convex opt

Stochastic Gradient Descent

Machine Learning Problems

- **Given data:**

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- **Learning a model's parameters:** $\underbrace{\sum_{i=1}^n \ell_i(w)}$

Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Big|_{w=w_t}$$

what if $n = 10^8$

$$= w_t - \eta \frac{1}{n} \sum_{i=1}^n \nabla_w \ell_i(w) \Big|_{w=w_t}$$

Machine Learning Problems

- **Given data:**

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- **Learning a model's parameters:** $\sum_{i=1}^n \ell_i(w)$

Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Bigg|_{w=w_t}$$

Stochastic Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w) \Big|_{w=w_t}$$

I_t drawn uniform at random from $\{1, \ldots, n\}$

$$\mathbb{E}[\nabla \ell_{I_t}(w)] = \sum_{i=1}^n \underbrace{\mathbb{P}(I_t = i)}_{= \frac{1}{n}} \nabla \ell_i(w) = \frac{1}{n} \sum_{i=1}^n \nabla \ell_i(w) =$$

Machine Learning Problems

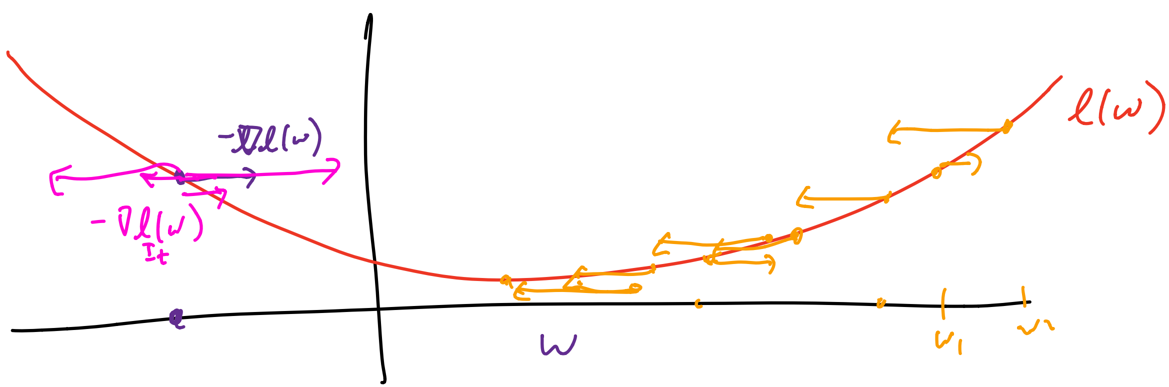
- **Learning a model's parameters:** $\sum_{i=1}^{n} \ell_i(w)$

Stochastic Gradient Descent:

$$\boxed{w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w)\Big|_{w=w_t}}$$

I_t drawn uniform at random from $\{1, \ldots, n\}$

$$\mathbb{E}[\nabla \ell_{I_t}(w)] = \nabla \ell(w)$$



$f(w_T) - f(w_*) \le \varepsilon$ flops to reach ε-good soln

Stochastic Gradient Descent

$n \log(1/\varepsilon)$ fewer iters

GD "typically" converges like e^{-T}

SGD "typically" converges like $\frac{1}{T}$ as $\frac{1}{\sqrt{T}}$

way smaller cost per iteration

indep. of n $\longrightarrow$ $1/\varepsilon$

Theorem

Let $\boxed{w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w)\Big|_{w=w_t}}$ I_t drawn uniform at random from $\{1,\dots,n\}$ so that

$$\mathbb{E}[\nabla \ell_{I_t}(w)] = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(w) =: \nabla \ell(w)$$

If $\|w_{\cancel{1}} - w_0\|_2^2 \le R$ and $\sup_w \max_i \|\nabla \ell_i(w)\|_2 \le G$ then

$$\mathbb{E}[\ell(\bar{w}) - \ell(w_*)] \le \frac{R}{2T\eta} + \frac{\eta G}{2} \le \sqrt{\frac{RG}{T}} \qquad \eta = \sqrt{\frac{R}{GT}}$$

$$\bar{w} = \frac{1}{T}\sum_{t=1}^{T} w_t$$

(In practice use last iterate)

Stochastic Gradient Descent

Proof

$$\mathbb{E}[||w_{t+1} - w_*||_2^2] = \mathbb{E}[||w_t - \eta \nabla \ell_{I_t}(w_t) - w_*||_2^2]$$

Stochastic Gradient Descent

Proof

$$\mathbb{E}[||w_{t+1} - w_*||_2^2] = \mathbb{E}[||w_t - \eta \nabla \ell_{I_t}(w_t) - w_*||_2^2]$$

$$= \mathbb{E}[||w_t - w_*||_2^2] - 2\eta \mathbb{E}[\nabla \ell_{I_t}(w_t)^T (w_t - w_*)] + \eta^2 \mathbb{E}[||\nabla \ell_{I_t}(w_t)||_2^2]$$

$$\leq \mathbb{E}[||w_t - w_*||_2^2] - 2\eta \mathbb{E}[\ell(w_t) - \ell(w_*)] + \eta^2 G$$

$$\mathbb{E}[\nabla \ell_{I_t}(w_t)^T (w_t - w_*)] = \mathbb{E}\Big[\mathbb{E}[\nabla \ell_{I_t}(w_t)^T (w_t - w_*) | I_1, w_1, \ldots, I_{t-1}, w_{t-1}]\Big]$$

$$= \mathbb{E}\Big[\nabla \ell(w_t)^T (w_t - w_*)\Big]$$

$$\geq \mathbb{E}\Big[\ell(w_t) - \ell(w_*)\Big]$$

convexity

$$\sum_{t=1}^{T} \mathbb{E}[\ell(w_t) - \ell(w_*)] \leq \frac{1}{2\eta}\Big(\mathbb{E}[||w_1 - w_*||_2^2] - \mathbb{E}[||w_{T+1} - w_*||_2^2] + T\eta^2 G\Big)$$

$$\leq \frac{R}{2\eta} + \frac{T\eta G}{2}$$

Stochastic Gradient Descent

Proof

Jensen's inequality:
For any random $Z \in \mathbb{R}^d$ and convex function $\phi : \mathbb{R}^d \to \mathbb{R}$, $\phi(\mathbb{E}[Z]) \leq \mathbb{E}[\phi(Z)]$

$$\mathbb{E}[\ell(\bar{w}) - \ell(w_*)] \leq \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\ell(w_t) - \ell(w_*)]$$

$$\bar{w} = \frac{1}{T}\sum_{t=1}^{T} w_t$$

Stochastic Gradient Descent

Proof

Jensen's inequality:
For any random $Z \in \mathbb{R}^d$ and convex function $\phi : \mathbb{R}^d \to \mathbb{R}$, $\phi(\mathbb{E}[Z]) \leq \mathbb{E}[\phi(Z)]$

$$\mathbb{E}[\ell(\bar{w}) - \ell(w_*)] \leq \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\ell(w_t) - \ell(w_*)]$$

$$\bar{w} = \frac{1}{T} \sum_{t=1}^{T} w_t$$

$$\boxed{\mathbb{E}[\ell(\bar{w}) - \ell(w_*)] \leq \frac{R}{2T\eta} + \frac{\eta G}{2} \leq \sqrt{\frac{RG}{T}}}$$

$$\eta = \sqrt{\frac{R}{GT}}$$

Mini-batch SGD

Instead of one iterate, average B stochastic gradient together

Advantages:

- de-noises gradient reducing variance of gradient
- Matrix computations ← Computing B gradients at a time
- Parallelization

compute gradients by sending a partition of the batch to each computer

Stochastic Gradient Descent: A Learning perspective

UNIVERSITY of WASHINGTON

Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset:
    - Sampled iid from some distribution p(**x**,**y**) on features:
  - Loss function, e.g., hinge loss, logistic loss,…
  - We often minimize loss in training data:

$$\ell_{\mathcal{D}}(\mathbf{w}) = \frac{1}{N}\sum_{j=1}^{N}\ell(\mathbf{w},\mathbf{x}^j)$$

- However, we should really minimize expected loss on all data:

$$\ell(\mathbf{w}) = E_{\mathbf{x}}\left[\ell(\mathbf{w},\mathbf{x})\right] = \int p(\mathbf{x})\ell(\mathbf{w},\mathbf{x})d\mathbf{x}$$

- So, we are approximating the integral by the average on the training data

Gradient descent in Terms of Expectations

> **"True" objective function:**

$$\ell(\mathbf{w}) = E_{\mathbf{x}}\left[\ell(\mathbf{w}, \mathbf{x})\right] = \int p(\mathbf{x})\ell(\mathbf{w}, \mathbf{x})d\mathbf{x}$$

> **Taking the gradient:**

> **"True" gradient descent rule:**

> **How do we estimate expected gradient?**

Warm up - Revisited

$nd + p + pd$

$w_{t+1} = w_t - \eta \nabla \left(y_{I_t} - h_{J_t}(x_{I_t})\right)^2$

$J_t \sim [p]$

$I_t \sim [n]$

1 float in NumPy = 8 bytes
$10^6 \approx 2^{20}$ bytes = 1 MB
$10^9 \approx 2^{30}$ bytes = 1 GB

```
# generate some nonsense data for an example
X = np.random.randn(n,d)
y = np.random.randn(n)
```

```
# generate the random features
G = np.random.randn(p, d)*np.sqrt(.1)
b = np.random.rand(p)*2*np.pi
```

```
# construct HTH
HTH = np.zeros((p,p))
HTy = np.zeros(p)
for i in range(n):
    hi = np.dot(X[i,:], G.T)+b
    HTH += np.outer(hi, hi)
    HTy += y[i]*hi
    if i % 1000==0: print(i)
```

```
H = np.dot(X, G.T) + b.T
HTH = np.dot(H.T, H)
HTy = np.dot(H.T, y)
```

```
# construct HTH
HTH = np.zeros((p,p))
HTy = np.zeros(p)
block = p
for i in range(int(np.ceil(n/block))+1):
    Hi = np.dot(X[i*block:min(n,(i+1)*block),:], G.T)+b
    HTH += np.dot(Hi.T, Hi)
    HTy += np.dot(Hi.T, y[i*block:min(n,(i+1)*block)])
```

```
w = np.linalg.solve(HTH + lam*np.eye(p), HTy)
```

For each block compute the memory required in terms of n, p, d.

If d << p << n, what is the most memory efficient program (blue, green, red)?

If you have unlimited memory, what do you think is the fastest program?