Warm up

```
# 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
```

```
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)
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)
```

```
# 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)
```

1 float in NumPy = 8 bytes

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

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

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?

Gradient Descent

Standard Machine Learning Problem Setup

- **Have a bunch of iid data:**

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

- **Want to learn a model's parameters:**

Each $\ell_i(w)$ is convex. $\qquad \sum_{i=1}^{n} \ell_i(w)$

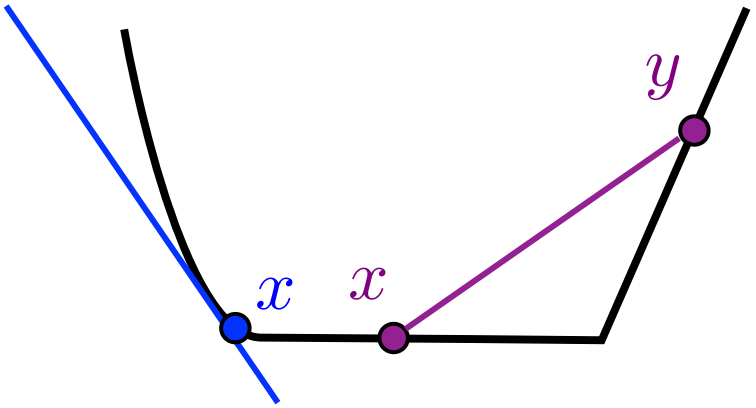
Machine Learning Problems

- **Have a bunch of iid data:**

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

- **Want to learn a model's parameters:**

Each $\ell_i(w)$ is convex. $\displaystyle\sum_{i=1}^{n} \ell_i(w)$



f convex:

$$f\left(\lambda x + (1-\lambda)y\right) \leq \lambda f(x) + (1-\lambda) f(y) \qquad \forall x, y, \lambda \in [0,1]$$

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \qquad \forall x, y$$

g is a subgradient at x if

$$f(y) \geq f(x) + g^T (y - x)$$

Taylor Series Approximation, 1-d

$$f(x+\delta) = f(x) + f'(x)\delta + \frac{1}{2}f''(x)\delta^2 + \dots$$

- **Gradient descent:**

Taylor Series Approximation, d dimensions

$$f(x+v) = f(x) + \nabla f(x)^T v + \tfrac{1}{2} v^T \nabla^2 f(x) v + \ldots$$

- **Gradient descent:**

Gradient Descent, LS

$$f(w) = \frac{1}{2}||\mathrm{X}w - \mathrm{y}||_2^2$$

$$\nabla f(w) = \mathbf{X}^T(\mathbf{X}w - \mathbf{y}) = \mathbf{X}^\mathbf{T}\mathbf{X}w - \mathbf{X}^T\mathbf{y}$$

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

$$= (I - \eta \mathbf{X}^T\mathbf{X})w_t + \eta \mathbf{X}^T\mathbf{y}$$

If, in round t, we ended up at w_*:

$$w_* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$(w_{t+1} - w_*) = (I - \eta \mathbf{X}^T\mathbf{X})(w_t - w_*) - \eta \mathbf{X}^T\mathbf{X}w_* + \eta \mathbf{X}^T\mathbf{y}$$

Gradient Descent, LS

$$f(w) = \frac{1}{2}||\mathrm{X}w - \mathrm{y}||_2^2$$

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

$$\begin{aligned}(w_{t+1} - w_*) &= (I - \eta \mathrm{X}^T\mathrm{X})(w_t - w_*) \\ &= (I - \eta \mathrm{X}^T\mathrm{X})^{t+1}(w_0 - w_*)\end{aligned}$$

Gradient Descent for Logistic Regression

Loss function: Conditional Likelihood

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

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \qquad P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

$$f(w) = \arg\min_w \sum_{i=1}^n \log(1 + \exp(-y_i\, x_i^T w))$$

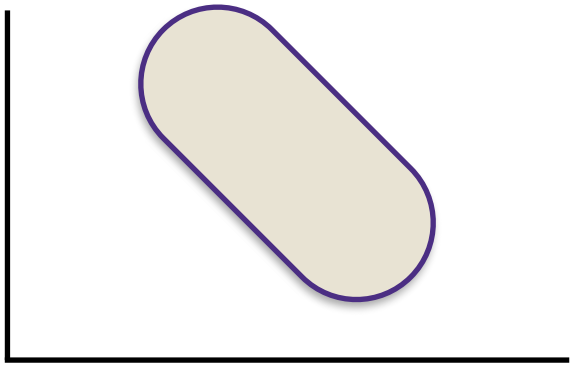
$$\nabla f(w) =$$

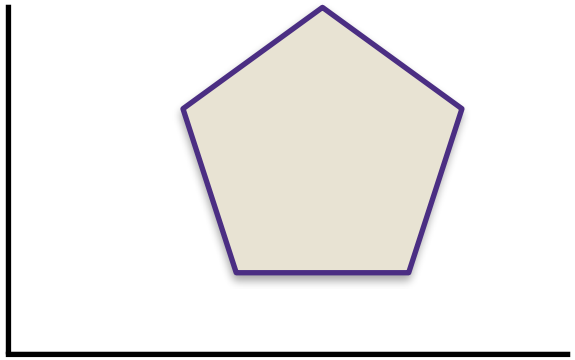
Convexity

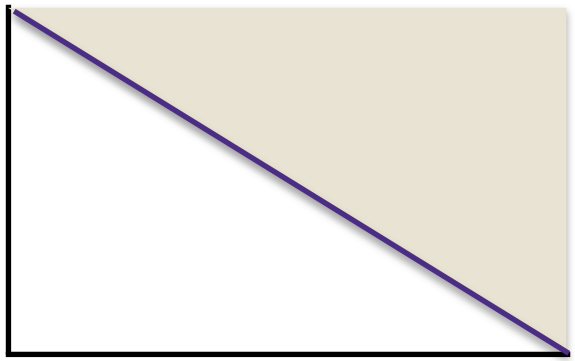
What is a convex set?

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

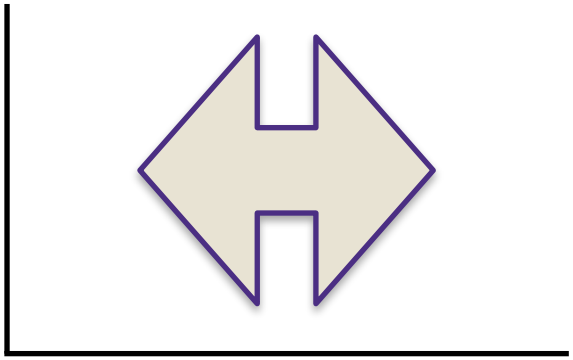
Examples of convex sets







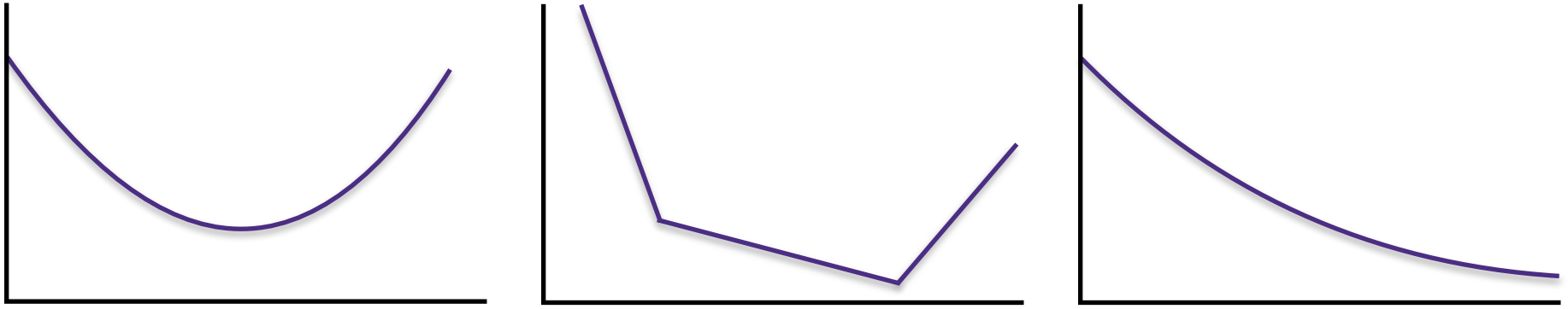
Examples of non-convex functions: anything else



What is a convex function?

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 $\lambda \in [0, 1]$

Examples of convex functions: "look like bowls"



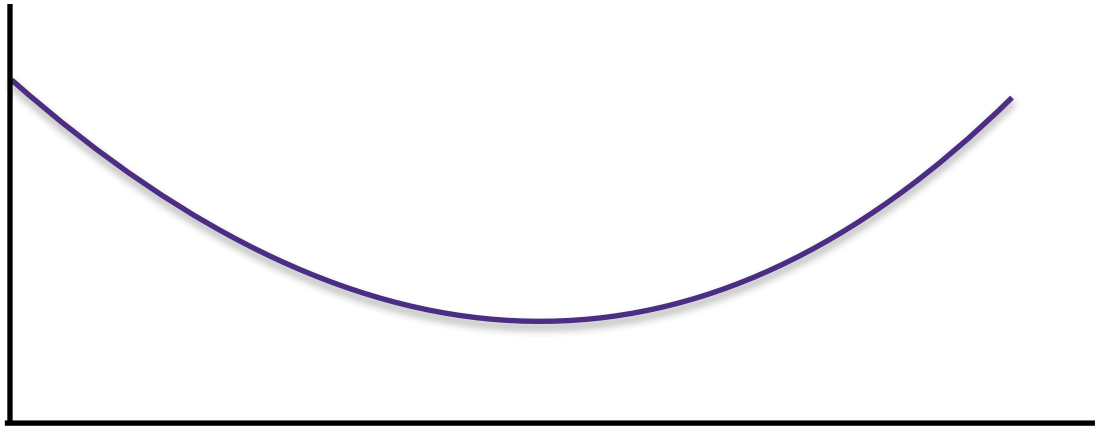
Examples of non-convex functions: anything else

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 $\{(x,t) \in \mathbb{R}^{d+1} : f(x) \leq t\}$ is convex

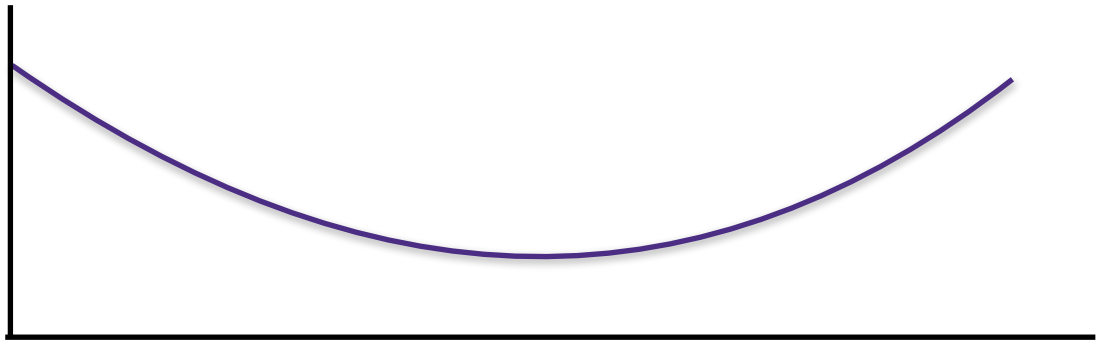


More definitions of convexity

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 $x, y \in dom(f)$



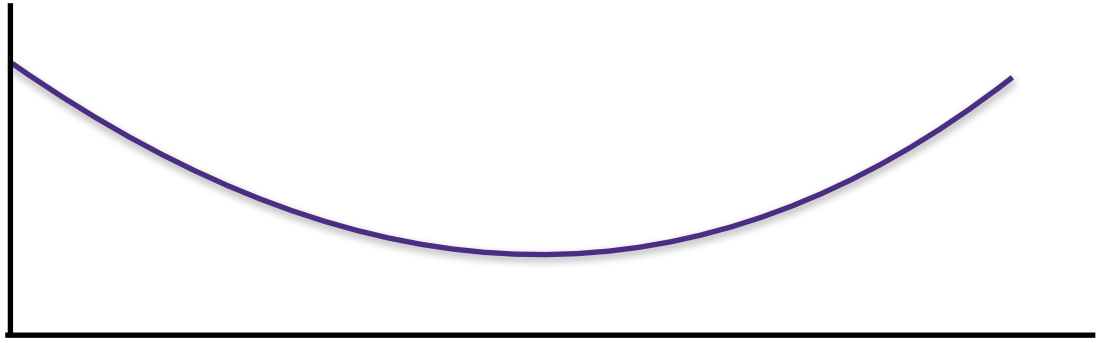
More definitions of convexity

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 $x, y \in dom(f)$

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 $x \in dom(f)$

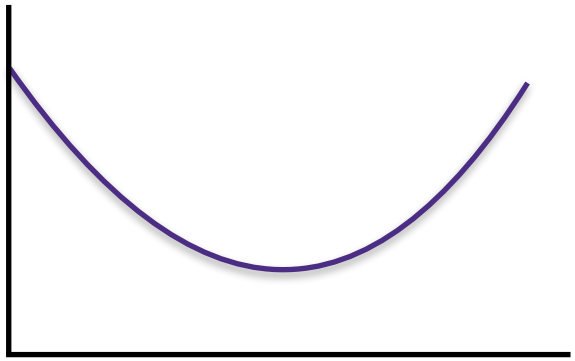


Why do we care about convexity?

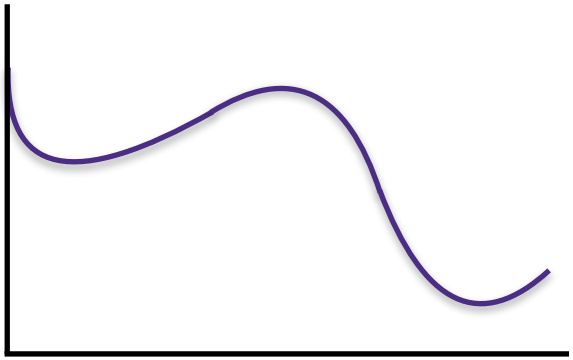
Convex functions

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

Convex Function



Non-convex Function



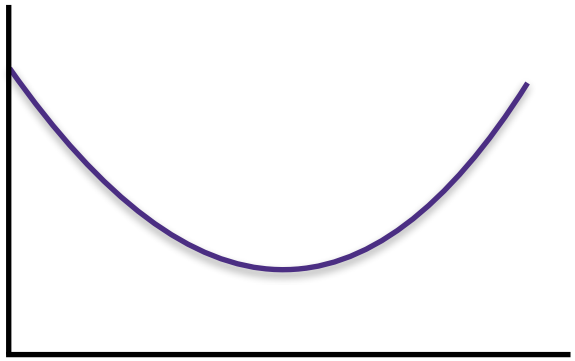
Gradient Descent

Initialize: $w_0 = 0$

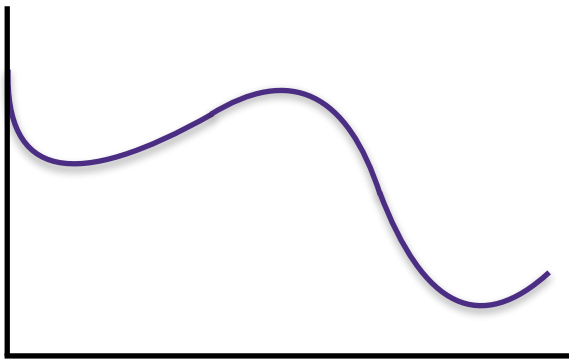
for $t = 1, 2, \ldots$

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

Convex Function



Non-convex Function



Sub-Gradient Descent

Initialize: $w_0 = 0$

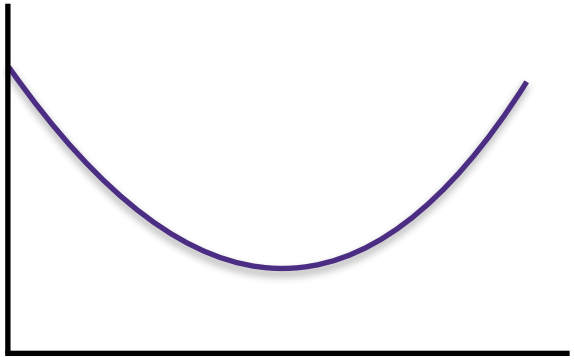
for $t = 1, 2, \ldots$

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

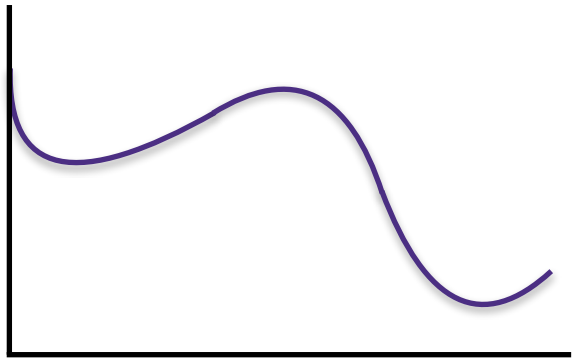
$w_{t+1} = w_t - \eta g_t$

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

Convex Function



Non-convex Function



Coordinate descent

Initialize: $w_0 = 0$

for $t = 1, 2, \ldots$

Let $i_t = t \ \% \ n$

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

Special case:

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:** $\sum_{i=1}^{n} \ell_i(w)$

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

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

Gradient Descent:

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

Optimization summary

- 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.)

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:** $\sum_{i=1}^{n} \ell_i(w)$

Gradient Descent:

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

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:

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

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 $\{1, \ldots, n\}$

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

Machine Learning Problems

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

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 $\{1, \ldots, n\}$

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

Stochastic Gradient Descent

Theorem

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

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

If $\|w_1 - w_0\|_2^2 \le R$ 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

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

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

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

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

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

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

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

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

Stochastic Gradient Descent

Proof

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

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

Stochastic Gradient Descent

Proof

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

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

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

Mini-batch SGD

Instead of one iterate, average B stochastic gradient together

Advantages:

- de-noises gradient
- Matrix computations
- Parallelization

Stochastic Gradient Descent: A Learning perspective

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

```
# 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)
```

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

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?