CPSC 540: Machine Learning

Stochastic Average Gradient and Kernels

Mark Schmidt

University of British Columbia

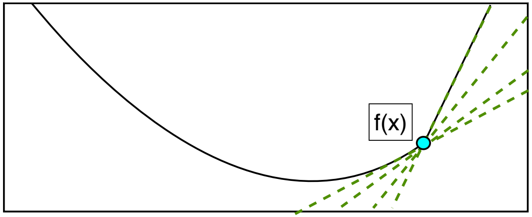
Winter 2016

Admin

- Assignment 2:
  - Due Tuesday
- Extra late Days:
  - To give possibility of two week-long extensions, allowing 4 late days.
  - But a maximum of 3 late days on any single assignment.
- Switch to Beamer?
  - Poll says?
  - Annotation vs. transition.

Last Time: Subgradients and Subgradient Method

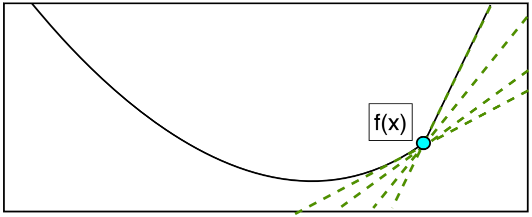
- Subgradients are a generalization of gradients for non-smooth optimization.
  - Slopes of linear underestimators, set of subgradients at x is sub-differential $\partial f(x)$.
  - If differentiable x, gradient is the only subgradient.



  - Subgradients exist everywhere for convex funcitons (except vertical asymptotes).

Last Time: Subgradients and Subgradient Method

- Subgradients are a generalization of gradients for non-smooth optimization.
  - Slopes of linear underestimators, set of subgradients at x is sub-differential $\partial f(x)$.
  - If differentiable x, gradient is the only subgradient.



  - Subgradients exist everywhere for convex funcitons (except vertical asymptotes).
  - We can define them locally for non-convex functions ("Clarke" subgradient).
- Subgradient method uses these to minimize a convex function:

$$x^{t+1} = x^t - \alpha_t g_t, \text{ where } g_t \in \partial f(x^t).$$

Last Time: Calculating Subgradients

- Computing general subgradient is complicated, but if f_1 and f_2 are convex then

$$\partial \max\{f_1(x), f_2(x)\} = \begin{cases} \nabla f_1(x) & f_1(x) > f_2(x) \\ \nabla f_2(x) & f_2(x) > f_1(x) \\ \theta \nabla f_1(x) + (1-\theta)\nabla f_2(x) & f_1(x) = f_2(x) \end{cases}$$

Last Time: Calculating Subgradients

- Computing general subgradient is complicated, but if f_1 and f_2 are convex then

$$\partial \max\{f_1(x), f_2(x)\} = \begin{cases} \nabla f_1(x) & f_1(x) > f_2(x) \\ \nabla f_2(x) & f_2(x) > f_1(x) \\ \theta \nabla f_1(x) + (1-\theta)\nabla f_2(x) & f_1(x) = f_2(x) \end{cases}$$

$$d \in \partial(f_1(x) + f_2(x)) \text{ if } d = d_1 + d_2 \text{ for } d_1 \in \partial f_1(x) \text{ and } d_2 \in \partial f_2(x).$$

Last Time: Calculating Subgradients

- Computing general subgradient is complicated, but if f_1 and f_2 are convex then

$$\partial \max\{f_1(x), f_2(x)\} = \begin{cases} \nabla f_1(x) & f_1(x) > f_2(x) \\ \nabla f_2(x) & f_2(x) > f_1(x) \\ \theta \nabla f_1(x) + (1-\theta)\nabla f_2(x) & f_1(x) = f_2(x) \end{cases}$$

$$d \in \partial(f_1(x) + f_2(x)) \text{ if } d = d_1 + d_2 \text{ for } d_1 \in \partial f_1(x) \text{ and } d_2 \in \partial f_2(x).$$

- So for SVMs,

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i)\} + \frac{\lambda}{2}\|w\|^2,$$

we can get a sub-gradient by computing

$$\frac{1}{n}\sum_{i=1}^{n} d_i + \lambda w, \text{ with } d_i = \begin{cases} -y_i x_i \geq 0 & \text{if } 1 - y_i(w^T x_i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

What is the best subgradient?

- We analyzed the subgradient method,

$$x^{t+1} = x^t - \alpha_t g_t, \text{ where } g_t \in \partial f(x^t),$$

under any choice of subgradient.

What is the best subgradient?

- We analyzed the subgradient method,

$$x^{t+1} = x^t - \alpha_t g_t, \text{ where } g_t \in \partial f(x^t),$$

  under any choice of subgradient.
- But what is the "best" subgradient to use?
  - Convex functions have directional derivatives everywhere.
  - Direction $-z^t$ that minimizes directional derivative is minimum-norm subgradient,

$$z^t = \mathop{\text{argmin}}_{z \in \partial f(x^t)} ||z||$$

  - This is the steepest descent direction for non-smooth convex optimization problems.

What is the best subgradient?

- We analyzed the subgradient method,

$$x^{t+1} = x^t - \alpha_t g_t, \text{ where } g_t \in \partial f(x^t),$$

  under any choice of subgradient.
- But what is the "best" subgradient to use?
  - Convex functions have directional derivatives everywhere.
  - Direction $-z^t$ that minimizes directional derivative is minimum-norm subgradient,

$$z^t = \underset{z \in \partial f(x^t)}{\text{argmin}} ||z||$$

  - This is the steepest descent direction for non-smooth convex optimization problems.
  - You can compute this for L1-regularization, but not many other problems.
  - Basis for best L1-regularization methods, combined (carefully) with Newton.

Last time: Stochastic sub-gradient

- We discussed minimizing finite sums,

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

when n is very large.

Last time: Stochastic sub-gradient

- We discussed minimizing finite sums,

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

  when n is very large.

- For non-smooth f_i, we discussed stochastic subgradient method,

$$x^{t+1} = x^t - \alpha g_{i_t},$$

  for some $g_{i_t} \in \partial f_{i_t}(x^t)$ for some random $i_t \in \{1, 2, \dots, n\}$.

Last time: Stochastic sub-gradient

- We discussed minimizing finite sums,

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

  when n is very large.
- For non-smooth f_i, we discussed stochastic subgradient method,

$$x^{t+1} = x^t - \alpha g_{i_t},$$

  for some $g_{i_t} \in \partial f_{i_t}(x^t)$ for some random $i_t \in \{1, 2, \dots, n\}$.
- Same convergence rate as deterministic subgradient method and n times faster.
- Suitable when d and n are both huge, with a *careful* implementation:
  - In high-level languages like Matlab, stochastic subgradient might be slow.
  - We have to carefully deal with sparsity of subgradients...

Stochastic Subgradient with Sparse Features

- For many datasets, our feature vectors x_i are very sparse:

"CPSC	"Expedia"	"vicodin"	<recipient name>	...
1	0	0	0	...
0	1	0	0	...
0	0	1	0	...
0	1	0	1	...
1	0	1	1	...

- Consider case where d is huge but each row x_i has at most k non-zeroes:
  - The $O(d)$ cost of stochastic subgradient might be too high.
  - We can often modify stochastic subgradient to have $O(k)$ cost.

Digression: Operations on Sparse Vectors

- Consider a vector $g \in \mathbb{R}^d$ with at most k non-zeroes:

$$g^T = \begin{bmatrix} 0 & 0 & 0 & 1 & 2 & 0 & -0.5 & 0 & 0 & 0 \end{bmatrix}.$$

- If $k << d$, we can store the vector using $O(k)$ storage instead of $O(d)$:
    - Store the non-zero values:

$$g_{\text{value}}^T = \begin{bmatrix} 1 & 2 & -0.5 \end{bmatrix}.$$

    - Store a pointer to where the non-zero values go:

$$g_{\text{point}}^T = \begin{bmatrix} 4 & 5 & 7 \end{bmatrix}.$$

- With this representation, we can do standard vector operations in $O(k)$:
    - Compute αg in $O(k)$ by computing αg_{value}.
    - For dense w, set $w = (w - g)$ in $O(k)$ by subracting g_{value} from w at positions g_{point}

Stochastic Subgradient with Sparse Features

- Consider optimizing the hinge-loss,

$$\underset{w\in\mathbb{R}^d}{\text{argmin}}\ \frac{1}{n}\sum_{i=1}^{n}\max\{0, 1 - y_i(w^T x_i)\},$$

when d is huge but each row has at most k non-zeroes.

Stochastic Subgradient with Sparse Features

- Consider optimizing the hinge-loss,

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i)\},$$

  when d is huge but each row has at most k non-zeroes.

- A stochastic subgradient method could use

$$w^{t+1} = w^t - \alpha_t g_{i_t}, \text{ where } g_i = \begin{cases} -y_i x_i & \text{if } 1 - y_i(w^T x_i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Stochastic Subgradient with Sparse Features

- Consider optimizing the hinge-loss,

$$\underset{w\in\mathbb{R}^d}{\text{argmin}}\ \frac{1}{n}\sum_{i=1}^{n}\max\{0, 1-y_i(w^Tx_i)\},$$

  when d is huge but each row has at most k non-zeroes.
- A stochastic subgradient method could use

$$w^{t+1} = w^t - \alpha_t g_{i_t},\ \text{where } g_i = \begin{cases} -y_ix_i & \text{if } 1-y_i(w^Tx_i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Notice that g_{i_t} has at most k non-zeroes:
  - Computing $\alpha_t g_{i_t}$ costs $O(k)$: multiply α_t by non-zeroes.
  - Computing $w^t - \alpha_t g_{i_t}$ costs $O(k)$: subtract non-zeroes.
- So stochastic subgradient is fast if k is small even if d is large.

Stochastic Subgradient with Sparse Features

- Consider the L2-regularized hinge-loss in the same setting,

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i)\} + \frac{\lambda}{2}\|w\|^2,$$

  using a stochastic subgradient method,

$$w^{t+1} = w^t - \alpha_t g_{i_t} - \alpha_t \lambda w^t, \text{ where } g_{i_t} \text{ is same as before}$$

Stochastic Subgradient with Sparse Features

- Consider the L2-regularized hinge-loss in the same setting,

$$\underset{w\in\mathbb{R}^d}{\text{argmin}} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i)\} + \frac{\lambda}{2}\|w\|^2,$$

  using a stochastic subgradient method,

$$w^{t+1} = w^t - \alpha_t g_{i_t} - \alpha_t \lambda w^t, \text{ where } g_{i_t} \text{ is same as before}$$

- While g_{i_t} has at most k non-zeros, w^t could have d non-zeroes:
  - So adding L2-regularization increases cost from $O(k)$ to $O(d)$?

Stochastic Subgradient with Sparse Features

- Consider the L2-regularized hinge-loss in the same setting,

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i)\} + \frac{\lambda}{2}\|w\|^2,$$

  using a stochastic subgradient method,

$$w^{t+1} = w^t - \alpha_t g_{i_t} - \alpha_t \lambda w^t, \text{ where } g_{i_t} \text{ is same as before}$$

- While g_{i_t} has at most k non-zeros, w^t could have d non-zeroes:
  - So adding L2-regularization increases cost from $O(k)$ to $O(d)$?
- To use L2-regularization and keep $O(k)$ cost, re-write iteration as

$$\begin{aligned} w^{t+1} &= w^t - \alpha_t g_{i_t} - \alpha_t \lambda w^t \\ &= \underbrace{(1 - \alpha_t\lambda)w^t}_{\text{changes scale of } w^t} - \underbrace{\alpha_t g_{i_t}}_{\text{sparse update}}. \end{aligned}$$

Stochastic Subgradient with Sparse Features

- Let's write the update as two steps

$$w^{t+\frac{1}{2}} = (1 - \alpha_t\lambda)w^t, \quad w^{t+1} = w^{t+\frac{1}{2}} - \alpha_t g_{i_t}.$$

Stochastic Subgradient with Sparse Features

- Let's write the update as two steps
$$w^{t+\frac{1}{2}} = (1 - \alpha_t\lambda)w^t, \quad w^{t+1} = w^{t+\frac{1}{2}} - \alpha_t g_{i_t}.$$
- We can implement both steps in $O(k)$ if we re-parameterize as
$$w^t = \beta^t v^t,$$
for some scalar β^t and vector v^t.

Stochastic Subgradient with Sparse Features

- Let's write the update as two steps

$$w^{t+\frac{1}{2}} = (1 - \alpha_t\lambda)w^t, \quad w^{t+1} = w^{t+\frac{1}{2}} - \alpha_t g_{i_t}.$$

- We can implement both steps in $O(k)$ if we re-parameterize as

$$w^t = \beta^t v^t,$$

for some scalar β^t and vector v^t.

- For the first step we need

$$\beta^{t+\frac{1}{2}} v^{t+\frac{1}{2}} = (1 - \alpha_t\lambda)\beta^t v^t,$$

which we can satisfy in $O(1)$ using $\beta^{t+\frac{1}{2}} = (1 - \alpha_t\lambda)\beta^t$ and $v^{t+\frac{1}{2}} = v^t$.

Stochastic Subgradient with Sparse Features

- Let's write the update as two steps

$$w^{t+\frac{1}{2}} = (1-\alpha_t\lambda)w^t, \quad w^{t+1} = w^{t+\frac{1}{2}} - \alpha_t g_{i_t}.$$

- We can implement both steps in $O(k)$ if we re-parameterize as

$$w^t = \beta^t v^t,$$

  for some scalar β^t and vector v^t.
- For the first step we need

$$\beta^{t+\frac{1}{2}} v^{t+\frac{1}{2}} = (1-\alpha_t\lambda)\beta^t v^t,$$

  which we can satisfy in $O(1)$ using $\beta^{t+\frac{1}{2}} = (1-\alpha_t\lambda)\beta^t$ and $v^{t+\frac{1}{2}} = v^t$.
- For the second step we need

$$\beta^{t+1} v^{t+1} = \beta^{t+\frac{1}{2}} v^{t+\frac{1}{2}} - \alpha_t g_{i_t}.$$

  which we can satisfy in $O(k)$ using $\beta^{t+1} = \beta^{t+\frac{1}{2}}$ and $v^{t+1} = v^{t+\frac{1}{2}} - \frac{\alpha_t}{\beta^{t+\frac{1}{2}}} g_{i_t}$.

Stochastic Subgradient with Sparse Features

- So we can implement the subgradient method with L2-regularization,

$$w^{t+1} = w^t - \alpha_t g_{i_t} - \alpha_t \lambda w^t,$$

in $O(k)$ by using the $w^t = \beta^t v^t$ representation and the update

$$\beta^{t+1} = (1 - \alpha_t \lambda)\beta^t, \quad v^{t+1} = v^t - \frac{\alpha_t}{\beta^{t+1}} g_{i_t}.$$

assuming that computing g_{i_t} can be done in $O(k)$ given β^t and v^t.

Stochastic Subgradient with Sparse Features

- So we can implement the subgradient method with L2-regularization,
$$w^{t+1} = w^t - \alpha_t g_{i_t} - \alpha_t \lambda w^t,$$
in $O(k)$ by using the $w^t = \beta^t v^t$ representation and the update
$$\beta^{t+1} = (1 - \alpha_t \lambda)\beta^t, \quad v^{t+1} = v^t - \frac{\alpha_t}{\beta^{t+1}} g_{i_t}.$$
assuming that computing g_{i_t} can be done in $O(k)$ given β^t and v^t.
- There exists efficient sparse updates in other scenarios too:
    - Duchi & Singer [2009]: L1-regularization proximal operator ("lazy updates").
    - Xu [2010]: L2-regularization and iterate average $\bar{w}^t$.

Stochastic Subgradient Methods in Practice

- Last time we argued that α_t must go to zero for convergence.
- Theory says using $\alpha_t = 1/\mu t$ and averaging is close to optimal:

Stochastic Subgradient Methods in Practice

- Last time we argued that α_t must go to zero for convergence.
- Theory says using $\alpha_t = 1/\mu t$ and averaging is close to optimal:
  - Except for some special cases, you should not do this.
    - Usually $\mu = O(1/n)$ or $O(1/\sqrt{n})$ so initial steps are huge.
    - Later steps are tiny: $1/t$ gets small very quickly.
    - Convergence rate slows dramatically if μ isn't accurate.
    - No adaptation to "easier" problems than worst case.

Stochastic Subgradient Methods in Practice

- Last time we argued that α_t must go to zero for convergence.
- Theory says using $\alpha_t = 1/\mu t$ and averaging is close to optimal:
  - Except for some special cases, you should not do this.
    - Usually $\mu = O(1/n)$ or $O(1/\sqrt{n})$ so initial steps are huge.
    - Later steps are tiny: $1/t$ gets small very quickly.
    - Convergence rate slows dramatically if μ isn't accurate.
    - No adaptation to "easier" problems than worst case.
- Tricks that can improve theoretical and practical properties:
  1. Use smaller initial step-sizes, that go to zero more slowly:
  $$\alpha_t = \gamma/\sqrt{t} \quad \text{or } \alpha_t = \gamma.$$
  2. Take a (weighted) average of the iterations or gradients:
  $$\bar{x}^t = \sum_{i=1}^{t} \omega_t z^t,$$
  where ω_t is weight at iteration t.

Stochastic Subgradient Methods in Practice

- Last time we argued that α_t must go to zero for convergence.
- Theory says using $\alpha_t = 1/\mu t$ and averaging is close to optimal:
  - Except for some special cases, you should not do this.
    - Usually $\mu = O(1/n)$ or $O(1/\sqrt{n})$ so initial steps are huge.
    - Later steps are tiny: $1/t$ gets small very quickly.
    - Convergence rate slows dramatically if μ isn't accurate.
    - No adaptation to "easier" problems than worst case.
- Tricks that can improve theoretical and practical properties:
  1. Use smaller initial step-sizes, that go to zero more slowly:
  $$\alpha_t = \gamma/\sqrt{t} \quad \text{or } \alpha_t = \gamma.$$
  2. Take a (weighted) average of the iterations or gradients:
  $$\bar{x}^t = \sum_{i=1}^{t} \omega_t z^t,$$
  where ω_t is weight at iteration t.
- These tricks usually help, but tuning is often required:
  - stochastic subgradient is not a black box.

Speeding up Stochastic Subgradient Methods

Results that support using large steps and averaging:

- Averaging later iterations achieves $O(1/t)$ in non-smooth case.
- Gradient averaging improves constants in analysis.
- $\alpha_t = O(1/t^\beta)$ for $\beta \in (0.5, 1)$ more robust than $\alpha_t = O(1/t)$.

Speeding up Stochastic Subgradient Methods

Results that support using large steps and averaging:

- Averaging later iterations achieves $O(1/t)$ in non-smooth case.
- Gradient averaging improves constants in analysis.
- $\alpha_t = O(1/t^\beta)$ for $\beta \in (0.5, 1)$ more robust than $\alpha_t = O(1/t)$.
- Constant step size $(\alpha_t = \alpha)$ achieves linear rate to accuracy $O(\alpha)$.
- In smooth case, iterate averaging is asymptotically optimal:
    - Achieves same rate as optimal stochastic Newton method.

Stochastic Newton Methods?

- Should we use Nesterov/Newton-like stochastic methods?
  - These do not improve the $O(1/\epsilon)$ convergence rate.

Stochastic Newton Methods?

- Should we use Nesterov/Newton-like stochastic methods?
  - These do not improve the $O(1/\epsilon)$ convergence rate.
- But some positive results exist.
  - Improves performance at start or if noise is small.
  - Newton-like AdaGrad method,

$$x^{t+1} = x^t + \alpha D \nabla f_{i_t}(x^t), \quad \text{with } D_{jj} = \sqrt{\sum_{k=1}^{t} \|\nabla_j f_{i_k}(x^t)\|}.$$

    - improves regret but not optimization error.
  - Two-phase Newton-like method achieves $O(1/\epsilon)$ without strong-convexity.

Stochastic Subgradient for Infinite Datasets?

- In analysis of stochastic subgradient, two assumptions on g_{i_t}:
  - Unbiased approximation of subgradient: $\mathbb{E}[g_{i_t}] = g_t$.
  - Variance is bounded: $\mathbb{E}[\|g_{i_t}\|^2] \leq B^2$.
- We can achieve this in the general setting:

$$\operatorname*{argmin}_{x \in \mathbb{R}^d} \mathbb{E}[f_i(x)].$$

Stochastic Subgradient for Infinite Datasets?

- In analysis of stochastic subgradient, two assumptions on g_{i_t}:
    - Unbiased approximation of subgradient: $\mathbb{E}[g_{i_t}] = g_t$.
    - Variance is bounded: $\mathbb{E}[\|g_{i_t}\|^2] \leq B^2$.
- We can achieve this in the general setting:

$$\underset{x \in \mathbb{R}^d}{\text{argmin}}\, \mathbb{E}[f_i(x)].$$

- We can use stochastic subgradient on IID samples from infinite dataset:
    - In this setting, we are directly optimizing test loss and cannot overfit.
    - We require $O(1/\epsilon)$ samples to reach test loss accuracy of ϵ (optimal?).

Stochastic Subgradient for Infinite Datasets?

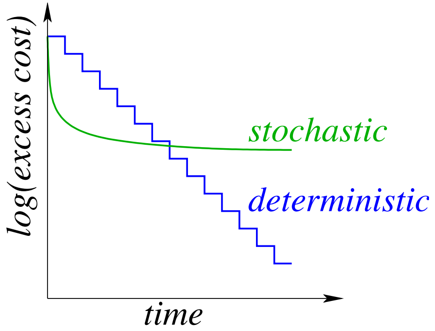
- In analysis of stochastic subgradient, two assumptions on g_{i_t}:
  - Unbiased approximation of subgradient: $\mathbb{E}[g_{i_t}] = g_t$.
  - Variance is bounded: $\mathbb{E}[\|g_{i_t}\|^2] \leq B^2$.
- We can achieve this in the general setting:

$$\operatorname*{argmin}_{x \in \mathbb{R}^d} \mathbb{E}[f_i(x)].$$

- We can use stochastic subgradient on IID samples from infinite dataset:
  - In this setting, we are directly optimizing test loss and cannot overfit.
  - We require $O(1/\epsilon)$ samples to reach test loss accuracy of ϵ (optimal?).
- Often used to justify doing one "pass" through data of stochastic subgradient:
  - If you only look at data point once, can be viewed as IID test sample.
  - Empirically, always worse than methods which do multiple passes.

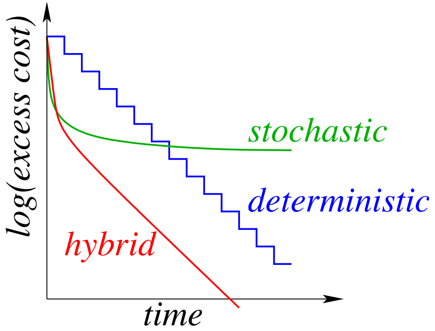
(pause)

Better Methods for Smooth Objectives and Finite Datasets?



- Stochastic methods:
  - $O(1/\epsilon)$ iterations but requires 1 gradient per iterations.
  - Rates are unimprovable for general stochastic objectives.
- Deterministic methods:
  - $O(\log(1/\epsilon)$ iterations but requires n gradients per iteration.
  - The faster rate is possible because n is finite.

Better Methods for Smooth Objectives and Finite Datasets?



- Stochastic methods:
  - $O(1/\epsilon)$ iterations but requires 1 gradient per iterations.
  - Rates are unimprovable for general stochastic objectives.
- Deterministic methods:
  - $O(\log(1/\epsilon)$ iterations but requires n gradients per iteration.
  - The faster rate is possible because n is finite.
- For finite n, can we design a better method?

Hybrid Deterministic-Stochastic

- Approach 1: control the sample size.

Hybrid Deterministic-Stochastic

- Approach 1: control the sample size.
- Deterministic method uses all n gradients,

$$\nabla f(x^t) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^t).$$

- Stochastic method approximates it with 1 sample,

$$\nabla f_{i_t}(x^t) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^t).$$

Hybrid Deterministic-Stochastic

- Approach 1: control the sample size.
- Deterministic method uses all n gradients,

$$\nabla f(x^t) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^t).$$

- Stochastic method approximates it with 1 sample,

$$\nabla f_{i_t}(x^t) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^t).$$

- A common variant is to use larger sample $\mathcal{B}^t$

$$\frac{1}{|\mathcal{B}^t|}\sum_{i\in\mathcal{B}^t} \nabla f_i(x^t) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^t),$$

particularly useful for vectorizaiton/parallelization.

Approach 1: Batching

- The SG method with a sample $\mathcal{B}^t$ uses iterations

$$x^{t+1} = x^t - \frac{\alpha^t}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t} f_i(x^t).$$

- For a fixed sample size $|\mathcal{B}^t|$, the rate is sublinear.

Approach 1: Batching

- The SG method with a sample $\mathcal{B}^t$ uses iterations

$$x^{t+1} = x^t - \frac{\alpha^t}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t} f_i(x^t).$$

- For a fixed sample size $|\mathcal{B}^t|$, the rate is sublinear.
- **Gradient error decreases as sample size $|\mathcal{B}^t|$ increases.**

Approach 1: Batching

- The SG method with a sample $\mathcal{B}^t$ uses iterations

$$x^{t+1} = x^t - \frac{\alpha^t}{|\mathcal{B}^t|}\sum_{i \in \mathcal{B}^t} f_i(x^t).$$

- For a fixed sample size $|\mathcal{B}^t|$, the rate is sublinear.
- **Gradient error decreases as sample size $|\mathcal{B}^t|$ increases.**
- Common to gradually increase the sample size $|\mathcal{B}^t|$.

Approach 1: Batching

- The SG method with a sample $\mathcal{B}^t$ uses iterations

$$x^{t+1} = x^t - \frac{\alpha^t}{|\mathcal{B}^t|}\sum_{i \in \mathcal{B}^t} f_i(x^t).$$

- For a fixed sample size $|\mathcal{B}^t|$, the rate is sublinear.
- **Gradient error decreases as sample size $|\mathcal{B}^t|$ increases.**
- Common to gradually increase the sample size $|\mathcal{B}^t|$.
- We can choose $|\mathcal{B}^t|$ to achieve a linear convergence rate:
    - Early iterations are cheap like SG iterations.
    - Later iterations can use a Newton-like method.

Approach 1: Batching

- The SG method with a sample $\mathcal{B}^t$ uses iterations

$$x^{t+1} = x^t - \frac{\alpha^t}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t} f_i(x^t).$$

- For a fixed sample size $|\mathcal{B}^t|$, the rate is sublinear.
- **Gradient error decreases as sample size $|\mathcal{B}^t|$ increases.**
- Common to gradually increase the sample size $|\mathcal{B}^t|$.
- We can choose $|\mathcal{B}^t|$ to achieve a linear convergence rate:
  - Early iterations are cheap like SG iterations.
  - Later iterations can use a Newton-like method.
- Another approach: at some point switch from stochastic to deterministic:
  - Often after a small number of passes.

Stochastic Average Gradient

- Growing $|\mathcal{B}^t|$ eventually requires O(n) iteration cost.
- **Can we have 1 gradient per iteration and only $O(\log(1/\epsilon))$ iterations?**

Stochastic Average Gradient

- Growing $|\mathcal{B}^t|$ eventually requires O(n) iteration cost.
- **Can we have 1 gradient per iteration and only $O(\log(1/\epsilon))$ iterations?**
  - YES!

Stochastic Average Gradient

- Growing $|\mathcal{B}^t|$ eventually requires O(n) iteration cost.
- **Can we have 1 gradient per iteration and only $O(\log(1/\epsilon))$ iterations?**
  - YES! The stochastic average gradient (SAG) algorithm:
    - Randomly select i_t from $\{1, 2, \ldots, n\}$ and compute $\nabla f_{i_t}(x^t)$.

$$x^{t+1} = x^t - \frac{\alpha^t}{n} \sum_{i=1}^{n} \nabla f_i(x^t)$$

Stochastic Average Gradient

- Growing $|\mathcal{B}^t|$ eventually requires O(n) iteration cost.
- **Can we have 1 gradient per iteration and only $O(\log(1/\epsilon))$ iterations?**
  - YES! The stochastic average gradient (SAG) algorithm:
    - Randomly select i_t from $\{1, 2, \ldots, n\}$ and compute $\nabla f_{i_t}(x^t)$.

$$x^{t+1} = x^t - \frac{\alpha^t}{n} \sum_{i=1}^{n} \nabla f_i(x^t)$$

Stochastic Average Gradient

- Growing $|\mathcal{B}^t|$ eventually requires O(n) iteration cost.
- **Can we have 1 gradient per iteration and only $O(\log(1/\epsilon))$ iterations?**
  - YES! The stochastic average gradient (SAG) algorithm:
    - Randomly select i_t from $\{1, 2, \ldots, n\}$ and compute $\nabla f_{i_t}(x^t)$.

$$x^{t+1} = x^t - \frac{\alpha^t}{n} \sum_{i=1}^{n} y_i^t$$

    - **Memory**: $y_i^t = \nabla f_i(x^t)$ from the last t where i was selected.

Stochastic Average Gradient

- Growing $|\mathcal{B}^t|$ eventually requires O(n) iteration cost.
- **Can we have 1 gradient per iteration and only $O(\log(1/\epsilon))$ iterations?**
  - YES! The stochastic average gradient (SAG) algorithm:
    - Randomly select i_t from $\{1, 2, \ldots, n\}$ and compute $\nabla f_{i_t}(x^t)$.

$$x^{t+1} = x^t - \frac{\alpha^t}{n} \sum_{i=1}^{n} y_i^t$$

    - **Memory**: $y_i^t = \nabla f_i(x^t)$ from the last t where i was selected.
    - Stochastic variant of earlier increment aggregated gradient (IAG).

Stochastic Average Gradient

- Growing $|\mathcal{B}^t|$ eventually requires O(n) iteration cost.
- **Can we have 1 gradient per iteration and only $O(\log(1/\epsilon))$ iterations?**
  - YES! The stochastic average gradient (SAG) algorithm:
    - Randomly select i_t from $\{1, 2, \dots, n\}$ and compute $\nabla f_{i_t}(x^t)$.

$$x^{t+1} = x^t - \frac{\alpha^t}{n} \sum_{i=1}^{n} y_i^t$$

    - **Memory**: $y_i^t = \nabla f_i(x^t)$ from the last t where i was selected.
    - Stochastic variant of earlier increment aggregated gradient (IAG).
  - Key idea: $y_i^t \to \nabla f_i(x^*)$ at the same time that $x^t \to x^*$:
    - So variance of the gradient approximation goes to 0.

SAG Algorithm

- Basic SAG algorithm (maintains $d = \sum_{i=1}^n y_i$):
    - Set $d = 0$ and gradient approximation $y_i = 0$ for $i = 1, 2, \dots, n$.
    - while(1)
        - Sample i from $\{1, 2, \dots, n\}$.
        - Compute $f_i'(x)$.
        - $d = d - y_i + f_i'(x)$.
        - $y_i = f_i'(x)$.
        - $x = x - \frac{\alpha}{n} d$.

SAG Algorithm

- Basic SAG algorithm (maintains $d = \sum_{i=1}^{n} y_i$):
    - Set $d = 0$ and gradient approximation $y_i = 0$ for $i = 1, 2, \ldots, n$.
    - while(1)
        - Sample i from $\{1, 2, \ldots, n\}$.
        - Compute $f_i'(x)$.
        - $d = d - y_i + f_i'(x)$.
        - $y_i = f_i'(x)$.
        - $x = x - \frac{\alpha}{n} d$.
- Iteration cost is $O(d)$, and "lazy updates" allows $O(k)$ with sparse gradients.
- For linear models where $f_i(w) = g(w^T x_i)$, then only require $O(n)$ memory:

$$\nabla f_i(w) = \underbrace{g'(w^T x_i)}_{\text{scalar}} \underbrace{x_i}_{\text{data}}.$$

Convergence Rate of SAG

If each f_i' is $L-$continuous and f is strongly-convex, with $\alpha_t = 1/16L$ SAG has

$$\mathbb{E}[f(x^t) - f(x^*)] \leqslant \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^t C,$$

Convergence Rate of SAG

If each f_i' is $L-$continuous and f is strongly-convex, with $\alpha_t = 1/16L$ SAG has

$$\mathbb{E}[f(x^t) - f(x^*)] \leqslant \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^t C,$$

where

$$C = [f(x^0) - f(x^*)] + \frac{4L}{n}\|x^0 - x^*\|^2 + \frac{\sigma^2}{16L}.$$

Convergence Rate of SAG

If each f_i' is $L-$continuous and f is strongly-convex, with $\alpha_t = 1/16L$ SAG has

$$\mathbb{E}[f(x^t) - f(x^*)] \leqslant \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^t C,$$

where

$$C = [f(x^0) - f(x^*)] + \frac{4L}{n}\|x^0 - x^*\|^2 + \frac{\sigma^2}{16L}.$$

- Number of f_i' evaluations to reach ϵ:
  - Stochastic: $O(\frac{L}{\mu}(1/\epsilon))$. (Best when n is enormous)

Convergence Rate of SAG

If each f_i' is $L-$continuous and f is strongly-convex, with $\alpha_t = 1/16L$ SAG has

$$\mathbb{E}[f(x^t) - f(x^*)] \leqslant \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^t C,$$

where

$$C = [f(x^0) - f(x^*)] + \frac{4L}{n}\|x^0 - x^*\|^2 + \frac{\sigma^2}{16L}.$$

- Number of f_i' evaluations to reach ϵ:
  - Stochastic: $O(\frac{L}{\mu}(1/\epsilon))$. (Best when n is enormous)
  - Gradient: $O(n\frac{L_f}{\mu}\log(1/\epsilon))$.

Convergence Rate of SAG

If each f_i' is $L-$continuous and f is strongly-convex, with $\alpha_t = 1/16L$ SAG has

$$\mathbb{E}[f(x^t) - f(x^*)] \leqslant \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^t C,$$

where

$$C = [f(x^0) - f(x^*)] + \frac{4L}{n}\|x^0 - x^*\|^2 + \frac{\sigma^2}{16L}.$$

- Number of f_i' evaluations to reach ϵ:
  - Stochastic: $O(\frac{L}{\mu}(1/\epsilon))$. (Best when n is enormous)
  - Gradient: $O(n\frac{L_f}{\mu}\log(1/\epsilon))$.
  - Nesterov: $O(n\sqrt{\frac{L_f}{\mu}}\log(1/\epsilon))$. (Best when n is small and L/μ is big)

Convergence Rate of SAG

If each f_i' is $L-$continuous and f is strongly-convex, with $\alpha_t = 1/16L$ SAG has

$$\mathbb{E}[f(x^t) - f(x^*)] \leqslant \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^t C,$$

where

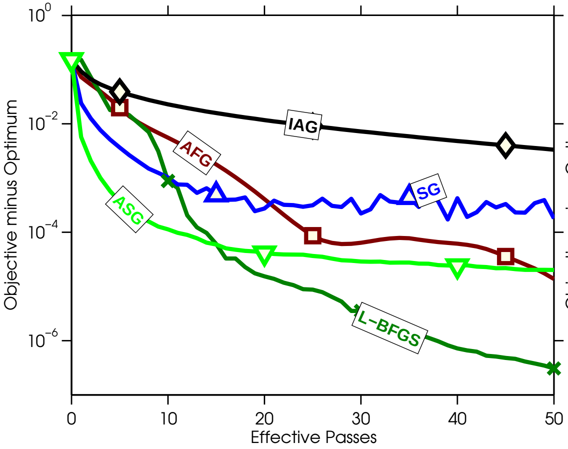
$$C = [f(x^0) - f(x^*)] + \frac{4L}{n}\|x^0 - x^*\|^2 + \frac{\sigma^2}{16L}.$$

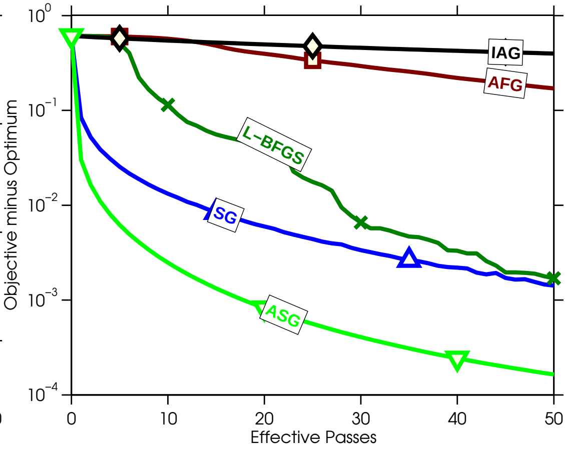
- Number of f_i' evaluations to reach ϵ:
  - Stochastic: $O(\frac{L}{\mu}(1/\epsilon))$. (Best when n is enormous)
  - Gradient: $O(n\frac{L_f}{\mu}\log(1/\epsilon))$.
  - Nesterov: $O(n\sqrt{\frac{L_f}{\mu}}\log(1/\epsilon))$. (Best when n is small and L/μ is big)
  - SAG: $O(\max\{n, \frac{L}{\mu}\}\log(1/\epsilon))$. (Best when n is big and L/μ is big)

(in this case $L_f \leq L$)

Comparing Deterministic and Stochastic Methods

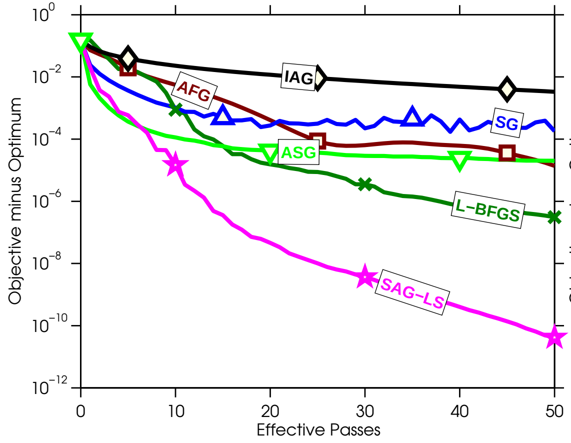
- Two benchmark logistic regression datasets:

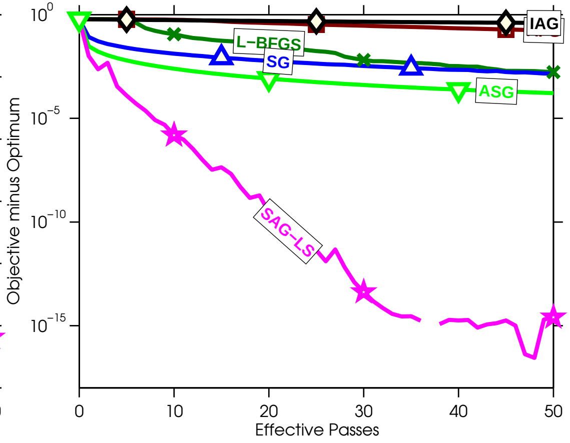




SAG Compared to Deterministic/Stochastic Methods

- Two benchmark logistic regression datasets:





Discussion of SAG and Beyond

- Implementation details (some backed up by theory, some not):
  - Common to use adaptive step-size procedure to estimate L.
  - Can use $\|x^{t+1} - x^t\|/\alpha = \frac{1}{n}d \approx \|\nabla f(x^t)\|$ to decide when to stop.
  - Lipschitz sampling of examples improves convergence rate:
    - As with coordinate descent, sample the ones that can change quickly more often.

Discussion of SAG and Beyond

- Implementation details (some backed up by theory, some not):
  - Common to use adaptive step-size procedure to estimate L.
  - Can use $\|x^{t+1} - x^t\|/\alpha = \frac{1}{n}d \approx \|\nabla f(x^t)\|$ to decide when to stop.
  - Lipschitz sampling of examples improves convergence rate:
    - As with coordinate descent, sample the ones that can change quickly more often.
- There are now a bunch of stochastic algorithm with $O(\log(1/\epsilon))$ rate:
  - SDCA, MISO, mixedGrad, SVRG, S2GD, Finito, SAGA, etc.
  - Proximal/accelerated/coordinate-wise/Newton-like versions.
- Some of these get rid of the memory...

Stochastic Variance-Reduced Gradient (SVRG)

SVRG algorithm: get rid of memory by occasionally computing exact gradient.

Stochastic Variance-Reduced Gradient (SVRG)

SVRG algorithm: get rid of memory by occasionally computing exact gradient.

- Start with x_0
- for $s = 0, 1, 2 \ldots$
    - $\nabla f(x_s) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x_s)$
    - $x^0 = x_s$

Stochastic Variance-Reduced Gradient (SVRG)

SVRG algorithm: get rid of memory by occasionally computing exact gradient.

- Start with x_0
- for $s = 0, 1, 2 \ldots$
    - $\nabla f(x_s) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x_s)$
    - $x^0 = x_s$
    - for $t = 0, 1, 2, \ldots m$
        - Randomly pick $i_t \in \{1, 2, \ldots, n\}$
        - $x^{t+1} = x^t - \alpha_t(\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x_s) + \nabla f(x_s))$.
    - $x_{s+1} = x^t$ for random $t \in \{0, 1, 2, \ldots, m\}$.

Stochastic Variance-Reduced Gradient (SVRG)

SVRG algorithm: get rid of memory by occasionally computing exact gradient.

- Start with x_0
- for $s = 0, 1, 2 \dots$
    - $\nabla f(x_s) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x_s)$
    - $x^0 = x_s$
    - for $t = 0, 1, 2, \dots m$
        - Randomly pick $i_t \in \{1, 2, \dots, n\}$
        - $x^{t+1} = x^t - \alpha_t(\nabla f_{i_t}(x^t) - \nabla f_{i_t}(x_s) + \nabla f(x_s))$.
    - $x_{s+1} = x^t$ for random $t \in \{0, 1, 2, \dots, m\}$.

Convergence properties similar to SAG (m large enough).

$O(d)$ storage at cost of 2 gradients per iteration and n gradients every $O(m)$ iterations.

(pause)

Motivation: Multi-Dimensional Polynomial Basis

- Recall using polynomial basis when we only have one features ($x_i \in \mathbb{R}$):

$$\hat{y}_i = \beta + w_1 x_i + w_2 x_i^2.$$

Motivation: Multi-Dimensional Polynomial Basis

- Recall using polynomial basis when we only have one features ($x_i \in \mathbb{R}$):

$$\hat{y}_i = \beta + w_1 x_i + w_2 x_i^2.$$

- We can fit these models using a change of basis:

$$\text{If } X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \text{ then let } \Phi(X) = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4^2) \end{bmatrix},$$

and L2-regulairzed least squares solution is

$$w = (\Phi(X)^T\Phi(X) + \lambda I)^{-1}\Phi(X)^T y.$$

Motivation: Multi-Dimensional Polynomial Basis

- Recall using polynomial basis when we only have one features ($x_i \in \mathbb{R}$):

$$\hat{y}_i = \beta + w_1 x_i + w_2 x_i^2.$$

- We can fit these models using a change of basis:

$$\text{If } X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \text{ then let } \Phi(X) = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4^2) \end{bmatrix},$$

and L2-regulairzed least squares solution is

$$w = (\Phi(X)^T \Phi(X) + \lambda I)^{-1} \Phi(X)^T y.$$

- How can we do this when we have a lot of features?

Motivation: Multi-Dimensional Polynomial Basis

- Approach 1: use polynomial basis for each variable:

$$X = \begin{bmatrix} 0.2 & 0.3 \\ 1 & 0.5 \\ -0.5 & -0.1 \end{bmatrix} \Rightarrow \Phi(X) = \begin{bmatrix} 1 & 0.2 & (0.2)^2 & 0.3 & (0.3)^2 \\ 1 & 1 & (1)^2 & 0.5 & (0.5)^2 \\ 1 & -0.5 & (-0.5)^2 & -0.1 & (-0.1)^2 \end{bmatrix}$$

Motivation: Multi-Dimensional Polynomial Basis

- Approach 1: use polynomial basis for each variable:

$$X = \begin{bmatrix} 0.2 & 0.3 \\ 1 & 0.5 \\ -0.5 & -0.1 \end{bmatrix} \Rightarrow \Phi(X) = \begin{bmatrix} 1 & 0.2 & (0.2)^2 & 0.3 & (0.3)^2 \\ 1 & 1 & (1)^2 & 0.5 & (0.5)^2 \\ 1 & -0.5 & (-0.5)^2 & -0.1 & (-0.1)^2 \end{bmatrix}$$

- But this is restrictve:
    - We should allow terms like $x_{i1}x_{i2}$ that depend on feature interactions.
    - But number of terms in X_{poly} would be huge:
        - Degree-5 polynomial basis has $O(d^5)$ terms:

$$x_{i1}^5, x_{i1}^4 x_{i2}, x_{i1}^4 x_{i3}, \dots, x_{i1}^3 x_{i2}^2, x_{i1}^3 x_{i2}^2, \dots, x_{i1}^3 x_{i2} x_{i3}, \dots$$

- If n is not too big, we can do this efficiently using the kernel trick.

Equivalent Form of Ridge Regression

- Recall the L2-regularized least squares model,

$$\underset{w \in \mathbb{R}^d}{\text{argmin}} \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

- We showed that the solution is

$$w = (\underbrace{X^T X}_{d \text{ by } d} + \lambda I_d)^{-1} X^T y,$$

where I_d is the d by d identity matrix.

Equivalent Form of Ridge Regression

- Recall the L2-regularized least squares model,

$$\underset{w\in\mathbb{R}^d}{\text{argmin}}\ \frac{1}{2}\|Xw-y\|^2+\frac{\lambda}{2}\|w\|^2.$$

- We showed that the solution is

$$w=(\underbrace{X^TX}_{d \text{ by } d}+\lambda I_d)^{-1}X^Ty,$$

  where I_d is the d by d identity matrix.
- An equivalent way to write the solution is:

$$w=X^T(\underbrace{XX^T}_{n \text{ by } n}+\lambda I_n)^{-1}y,$$

  by using a variant of the matrix inversion lemma.
- Computing w with this formula is faster if $n << d$:
  - since XX^T is n by n while X^TX is d by d.

Predictions using Equivalent Form

- Given test data $\hat{X}$, we predict $\hat{y}$ using:

$$\begin{aligned}\hat{y} &= \hat{X}w \\ &= \hat{X}X^T(XX^T + \lambda I_n)^{-1}y\end{aligned}$$

Predictions using Equivalent Form

- Given test data $\hat{X}$, we predict $\hat{y}$ using:

$$\begin{aligned}\hat{y} &= \hat{X}w \\ &= \hat{X}X^T(XX^T + \lambda I_n)^{-1}y\end{aligned}$$

- If we define $K = XX^T$ (Gram matrix) and $\hat{K} = \hat{X}X^T$, then we have

$$\hat{y} = \hat{K}(K + \lambda I_n)^{-1}y.$$

- Key observation behind kernel trick:
  - If we have the K and $\hat{K}$, we don't need the features.

Gram Matrix

- The Gram matrix K is defined by:

$$K = XX^T = \begin{bmatrix} — & x_1 & — \\ — & x_2 & — \\ & \vdots & \\ — & x_n & — \end{bmatrix} \begin{bmatrix} | & | & | \\ x_1 & x_2 & x_3 \\ | & | & | \end{bmatrix}$$

$$= \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & \cdots & x_1^T x_n \\ x_2^T x_1 & x_2^T x_2 & \cdots & x_2^T x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n^T x_1 & x_n^T x_2 & \cdots & x_n^T x_n \end{bmatrix}$$

- K contains the inner products between all training examples.

Gram Matrix

- The Gram matrix K is defined by:

$$K = XX^T = \begin{bmatrix} — & x_1 & — \\ — & x_2 & — \\ & \vdots & \\ — & x_n & — \end{bmatrix} \begin{bmatrix} | & | & | \\ x_1 & x_2 & x_3 \\ | & | & | \end{bmatrix}$$

$$= \begin{bmatrix} x_1^Tx_1 & x_1^Tx_2 & \cdots & x_1^Tx_n \\ x_2^Tx_1 & x_2^Tx_2 & \cdots & x_2^Tx_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n^Tx_1 & x_n^Tx_2 & \cdots & x_n^Tx_n \end{bmatrix}$$

- K contains the inner products between all training examples.
- $\hat{K}$ contains the inner products between training and test examples.
  - If we can compute inner products $k(x_i, x_j) = x_i^Tx_j$, we don't need x_i and x_j.

Polynomial Kernel

- Consider two examples x_i and x_j for a two-dimensional dataset:

$$x_i = (x_{i1}, x_{i2}), \quad x_j = (x_{j1}, x_{j2}).$$

- Consider a particular degree-2 basis ϕ:

$$\phi(x_i) = (x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2).$$

Polynomial Kernel

- Consider two examples x_i and x_j for a two-dimensional dataset:

$$x_i = (x_{i1}, x_{i2}), \quad x_j = (x_{j1}, x_{j2}).$$

- Consider a particular degree-2 basis ϕ:

$$\phi(x_i) = (x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2).$$

- We can compute inner product $\phi(x_i)^T\phi(x_j)$ without forming $\phi(x_i)$ and $\phi(x_j)$,

Polynomial Kernel

- Consider two examples x_i and x_j for a two-dimensional dataset:

$$x_i = (x_{i1}, x_{i2}), \quad x_j = (x_{j1}, x_{j2}).$$

- Consider a particular degree-2 basis ϕ:

$$\phi(x_i) = (x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2).$$

- We can compute inner product $\phi(x_i)^T\phi(x_j)$ without forming $\phi(x_i)$ and $\phi(x_j)$,

$$\begin{aligned}
\phi(x_i)^T\phi(x_j) &= \begin{bmatrix} x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix} \phi(x_j) \\
&= x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2 \\
&= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 && \text{(completing the square)} \\
&= \left(\sum_{k=1}^d x_{ik}x_{jk}\right)^2 \\
&= (x_i^T x_j)^2.
\end{aligned}$$

Polynomial Kernel with Higher Degrees

- If we want all degree-4 "monomials", raise to 4^{th} power:

$$\phi(x_i)^T\phi(x_j) = (x_i^Tx_j)^4,$$

where $\phi(x_i)$ is weighted version of $x_{i1}^4, x_{i1}^3x_{i2}, x_{i1}^2x_{i2}^2, x_{i1}x_{i2}^3, x_{i2}^4$.

Polynomial Kernel with Higher Degrees

- If we want all degree-4 “monomials”, raise to 4^{th} power:

$$\phi(x_i)^T\phi(x_j) = (x_i^T x_j)^4,$$

where $\phi(x_i)$ is weighted version of $x_{i1}^4, x_{i1}^3x_{i2}, x_{i1}^2x_{i2}^2, x_{i1}x_{i2}^3, x_{i2}^4$.

- If you want bias or lower-order terms like x_{i1}, add constant inside power:

$$(1 + x_i^T x_j)^2 = 1 + 2x_i^T x_j + (x_i^T x_j)^2$$

$$= \begin{bmatrix} 1 & 2x_{i1} & 2x_{i2} & x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2} \end{bmatrix} \begin{bmatrix} 1 \\ 2x_{j1} \\ 2x_{j2} \\ x_{j1}^2 \\ \sqrt{2}x_{j1}x_{j2} \\ x_{j2} \end{bmatrix} = \phi(x_i)^T\phi(x_j),$$

- These formulas still work for any dimension of the x_i.

Kernel Trick

- Using polynomial basis of degree 'p' with the kernel trick:
  - Compute K and $\hat{K}$ which have elements:

$$k(x_i, x_j) = (1 + x_i^T x_j)^p, \quad \hat{k}(\hat{x}_i, x_j) = (1 + \hat{x}_i^T x_j)^p.$$

  - Make predictions using:

$$\hat{y} = \hat{K}(K + \lambda I)^{-1} y.$$

Kernel Trick

- Using polynomial basis of degree ‘p’ with the kernel trick:
  - Compute K and $\hat{K}$ which have elements:

$$k(x_i, x_j) = (1 + x_i^T x_j)^p, \quad \hat{k}(\hat{x}_i, x_j) = (1 + \hat{x}_i^T x_j)^p.$$

  - Make predictions using:

$$\hat{y} = \hat{K}(K + \lambda I)^{-1} y.$$

  - Cost is $O(n^2 d + n^3)$ even though number of features is $O(d^p)$.

Kernel Trick

- Using polynomial basis of degree ‘p’ with the kernel trick:
  - Compute K and $\hat{K}$ which have elements:

$$k(x_i, x_j) = (1 + x_i^T x_j)^p, \quad \hat{k}(\hat{x}_i, x_j) = (1 + \hat{x}_i^T x_j)^p.$$

  - Make predictions using:

$$\hat{y} = \hat{K}(K + \lambda I)^{-1} y.$$

  - Cost is $O(n^2 d + n^3)$ even though number of features is $O(d^p)$.
- Kernel trick lets us fit regression models without explicit features:
  - We can interpret $k(i, j)$ as a “similarity” measure between objects.
  - We can apply regression to data where we don’t know features but have “similarity”.
  -
    - “String” kernels, “graph” kernels, “image” kernels, etc.

Summary

- Stochastic subgradient methods:
  - Tricks like $\beta^t v^t$ allow training on huge sparse datasets.
  - Different step-size strategies and averaging significantly improve performance.
  - Algorithm works with infinite training examples.

Summary

- Stochastic subgradient methods:
  - Tricks like $\beta^t v^t$ allow training on huge sparse datasets.
  - Different step-size strategies and averaging significantly improve performance.
  - Algorithm works with infinite training examples.
- Stochastic average gradient: $O(\log(1/\epsilon))$ iterations with 1 gradient per iteration.

Summary

- Stochastic subgradient methods:
  - Tricks like $\beta^t v^t$ allow training on huge sparse datasets.
  - Different step-size strategies and averaging significantly improve performance.
  - Algorithm works with infinite training examples.
- Stochastic average gradient: $O(\log(1/\epsilon))$ iterations with 1 gradient per iteration.
- Kernel trick: allows working with “similarity” instead of features.

- Next time: how to make/use kernels, and we start unsuperivsed learning.