Announcements

- Project proposal due next week: Tuesday 10/24
- Still looking for people to work on deep learning Phytolith project, join #phytolith slack channel

Gradient Descent

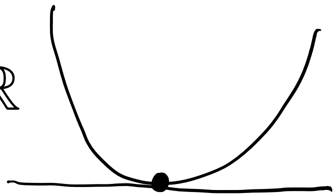
Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 16, 2016

Machine Learning Problems

- Have a bunch of iid data of the form:

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

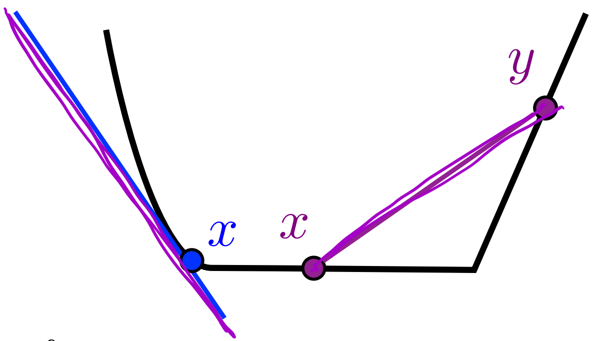


- Learning a model's parameters:

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

Each $\ell_i(w)$ is convex.

f(x) is a minimum of f if g is a sub-gradient at x



g is a subgradient at x if

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

f convex:

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

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

Machine Learning Problems

- Have a bunch of iid data of the form:

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

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

  Each $\ell_i(w)$ is convex.

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

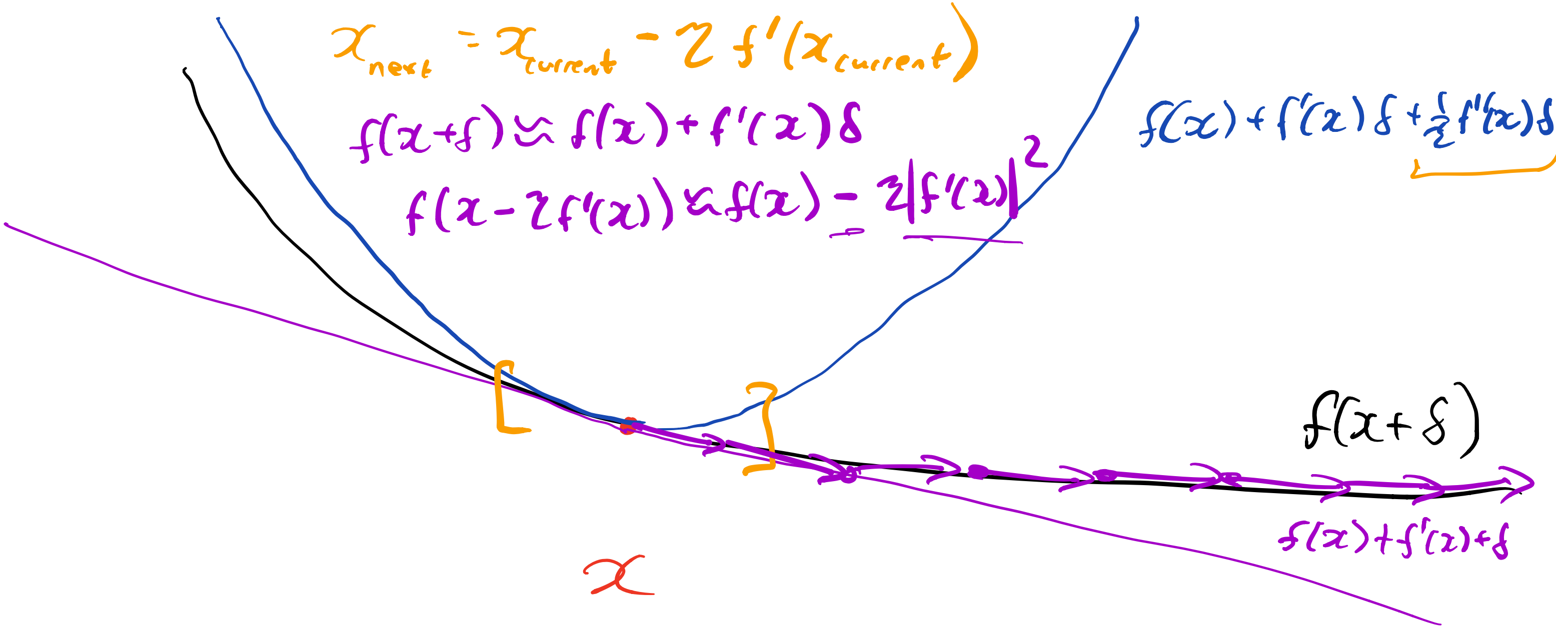
Squared error Loss: $\ell_i(w) = (y_i - x_i^T w)^2$

Taylor Series Approximation

- Taylor series in one dimension:

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

- Gradient descent:



Taylor Series Approximation

PSD

f is convex $\iff \nabla^2 f(x) \succeq 0 \;\; \forall x$

- Taylor series in **d** dimensions:

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

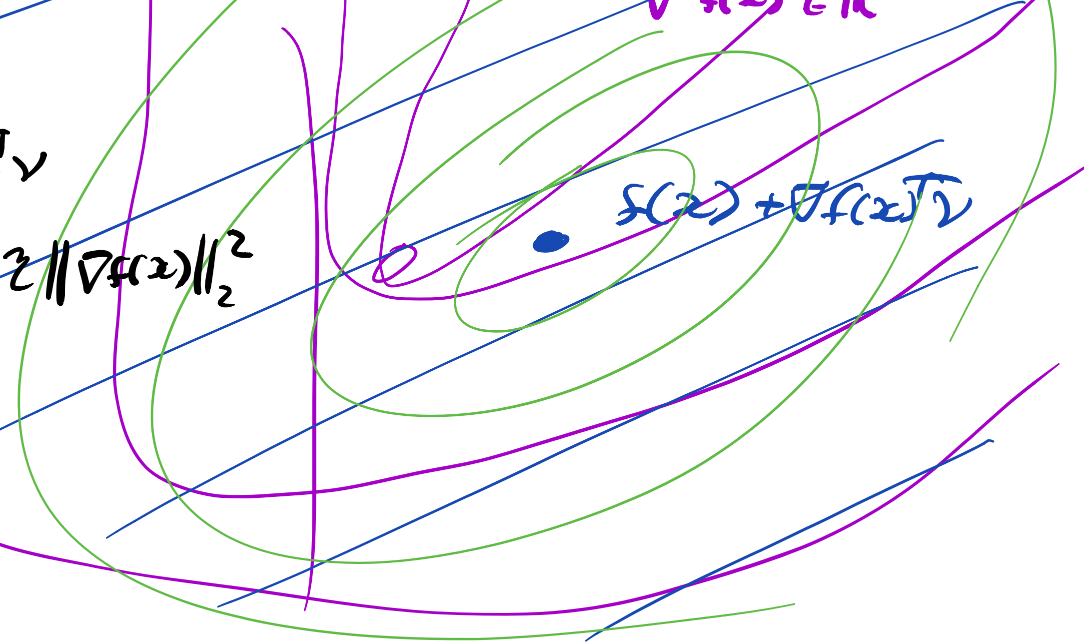
Hessian of f at x

$\nabla^2 f(x) \in \mathbb{R}^{d \times d}$

- Gradient descent:

$$f(x+v) \simeq f(x) + \nabla f(x)^T v$$

$$f(x - \eta \nabla f(x)) \simeq f(x) - \eta \|\nabla f(x)\|_2^2$$



General case

In general for Newton's method to achieve $f(w_t) - f(w_*) \leq \epsilon$:

$$t \simeq O(\log(\log(1/\epsilon)))$$

So why are ML problems overwhelmingly solved by gradient methods?

Hint: v_t is solution to : $\nabla^2 f(w_t) v_t = -\nabla f(w_t)$

General Convex case $f(w_t) - f(w_*) \leq \epsilon$

Clean convergence proofs: Bubeck

Newton's method:

$$t \approx \log(\log(1/\epsilon))$$

Gradient descent:

- f is *smooth* and *strongly convex:* $aI \preceq \nabla^2 f(w) \preceq bI$
- f is *smooth*: $\nabla^2 f(w) \preceq bI$
- f is potentially non-differentiable: $||\nabla f(w)||_2 \leq c$

Nocedal +Wright, Bubeck

Other: BFGS, Heavy-ball, BCD, SVRG, ADAM, Adagrad,…

Revisiting… Logistic Regression

Machine Learning – CSE546

Kevin Jamieson
University of Washington

October 16, 2016

Loss function: Conditional Likelihood

- Have a bunch of iid data of the form: $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

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

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

$$\nabla f(w) =$$

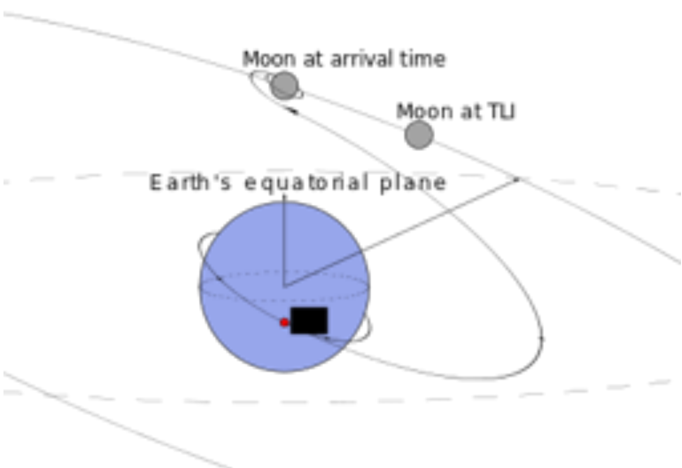
Online Learning

Machine Learning – CSE546

Kevin Jamieson
University of Washington

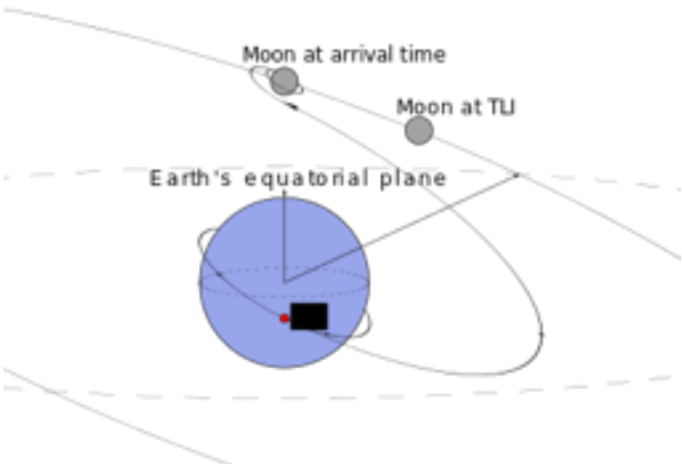
October 18, 2016

Going to the moon



Guidance computer predicts trajectories around moon and back with
- Noisy sensors
- Imperfect models
- Little computational power
- Big risk of failure

Going to the moon



Guidance computer predicts trajectories around moon and back with

- Noisy sensors
- Imperfect models
- Little computational power
- Big risk of failure

Why is Tom Hanks flying erratically?

Because they didn't have the power to turn on the Kalman FIlter!

State Estimation

- Predict current state given past state and current control input

$$\widetilde{w}_n = f(w_{n-1}) + g(u_n)$$

- Given current context, x_n compare your prediction to noisy measurement y_n

$$\ell_n(\widetilde{w}_n) = (y_n - h(x_n, \widetilde{w}_n))^2$$

- Update current state to include measurement

$$w_n = \widetilde{w}_n - K_n \nabla_w \ell_n(w)\Big|_{w=\widetilde{w}_n}$$

Kalman filter does optimal least squares state estimation if f, g, h are linear!

Recursive Least Squares (RLS)

Least squares = special case of Kalman Filter: no dynamics, no control

$$\widetilde{w}_n = f(w_{n-1}) + g(u_n)$$

$$\ell_n(\widetilde{w}_n) = (y_n - h(x_n, \widetilde{w}_n))^2$$

$$w_n = \widetilde{w}_n - K_n \nabla_w \ell_n(w)\Big|_{w=\widetilde{w}_n}$$

Recursive Least Squares (RLS)

Least squares = special case of Kalman Filter: no dynamics, no control

$$\begin{aligned}\widetilde{w}_n &= f(w_{n-1}) + g(u_n) \\ &= w_{n-1}\end{aligned}$$

$$\begin{aligned}\ell_n(\widetilde{w}_n) &= (y_n - h(x_n, \widetilde{w}_n))^2 \\ &= (y_n - x_n^T \widetilde{w}_n)^2 \\ &= (y_n - x_n^T w_{n-1})^2\end{aligned}$$

Ideally:

$$w_n = \arg\min_w \sum_{i=1}^{n} (y_i - x_i^T w)^2$$

$$\begin{aligned}w_n &= \widetilde{w}_n - K_n \nabla_w \ell_n(w)\Big|_{w=\widetilde{w}_n} \\ &= w_{n-1} + 2(y_n - x_n^T w_{n-1}) K_n x_n\end{aligned}$$

Recursive Least Squares (RLS)

Sherman–Morrison: $$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u}.$$

$$w_n = \left(\sum_{i=1}^n x_i x_i^T \right)^{-1} \sum_{i=1}^n x_i y_i$$

Ideally:

$$w_n = \arg\min_w \sum_{i=1}^n (y_i - x_i^T w)^2$$

Recursive Least Squares (RLS)

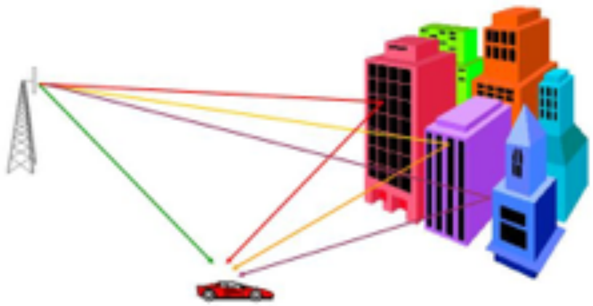
$$w_n = \left(\sum_{i=1}^n x_i x_i^T \right)^{-1} \sum_{i=1}^n x_i y_i$$

Great, what's the time-complexity of this?

It is 2017. Not the 60's… is limited computation still really a problem?

Digital Signal Processing

The original "Big Data"

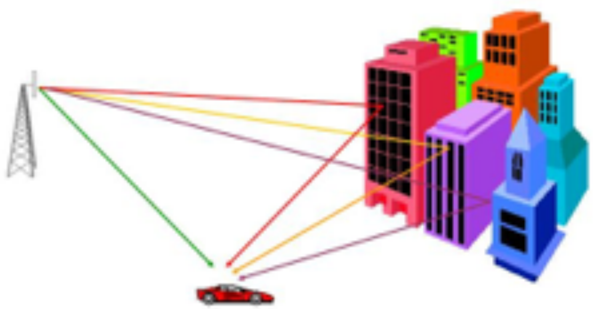


Wifi/cell-phones are *constantly* solving least squares to invert out multipath

Low power devices, high data rates

Digital Signal Processing

The original "Big Data"



Wifi/cell-phones are *constantly* solving least squares to invert out multipath

Low power devices, high data rates

Gigabytes of data per second

Incremental Gradient Descent

(x_t, y_t) arrive:

Note: no matrix multiply

$$w_{t+1} = w_t - \eta \left[\nabla_w (y_t - x_t^T w)^2 \Big|_{w=w_t} \right]$$

We know RLS is exact. How much worse is this?

In general convex $\ell_t(w)$ arrives:

$$\ell(\cdot) \text{ is convex} \iff \ell(y) \geq \ell(x) + \nabla \ell(x)^T (y - x) \ \forall x, y$$

Incremental Gradient Descent

$$||w_{t+1} - w_*||_2^2 = ||w_t - \eta \nabla \ell_t(w_t) - w_*||_2^2$$

Incremental Gradient Descent

Stochastic Gradient Descent

- Have a bunch of iid data of the form:

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

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

  Each $\ell_i(w)$ is convex.

Stochastic Gradient Descent

- Have a bunch of iid data of the form:

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

- Learning a model's parameters:

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

Each $\ell_i(w)$ is convex.

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

- Have a bunch of iid data of the form:

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

- Learning a model's parameters:

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

Each $\ell_i(w)$ is convex.

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 $\{1, \dots, n\}$

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

Stochastic Gradient Descent

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

Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:

$$E_{\mathbf{x}}\left[\ell(\mathbf{w},\mathbf{x})\right] = E_{\mathbf{x}}\left[\ln P(y|\mathbf{x},\mathbf{w}) - \lambda||\mathbf{w}||_2^2\right]$$

- Batch gradient ascent updates:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta\left\{-\lambda w_i^{(t)} + \frac{1}{N}\sum_{j=1}^{N} x_i^{(j)}[y^{(j)} - P(Y=1|\mathbf{x}^{(j)},\mathbf{w}^{(t)})]\right\}$$

- Stochastic gradient ascent updates:
  - Online setting:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t\left\{-\lambda w_i^{(t)} + x_i^{(t)}[y^{(t)} - P(Y=1|\mathbf{x}^{(t)},\mathbf{w}^{(t)})]\right\}$$

Stochastic Gradient Descent: A Learning perspective

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 16, 2016

Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset:
    - Sampled iid from some distribution p(**x**) 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?

SGD: Stochastic Gradient Descent

- "True" gradient: $$\nabla \ell(\mathbf{w}) = E_{\mathbf{x}}\left[\nabla \ell(\mathbf{w}, \mathbf{x})\right]$$

- Sample based approximation:

- What if we estimate gradient with just one sample???
  - Unbiased estimate of gradient
  - Very noisy!
  - Called stochastic gradient descent
    - Among many other names
  - VERY useful in practice!!!