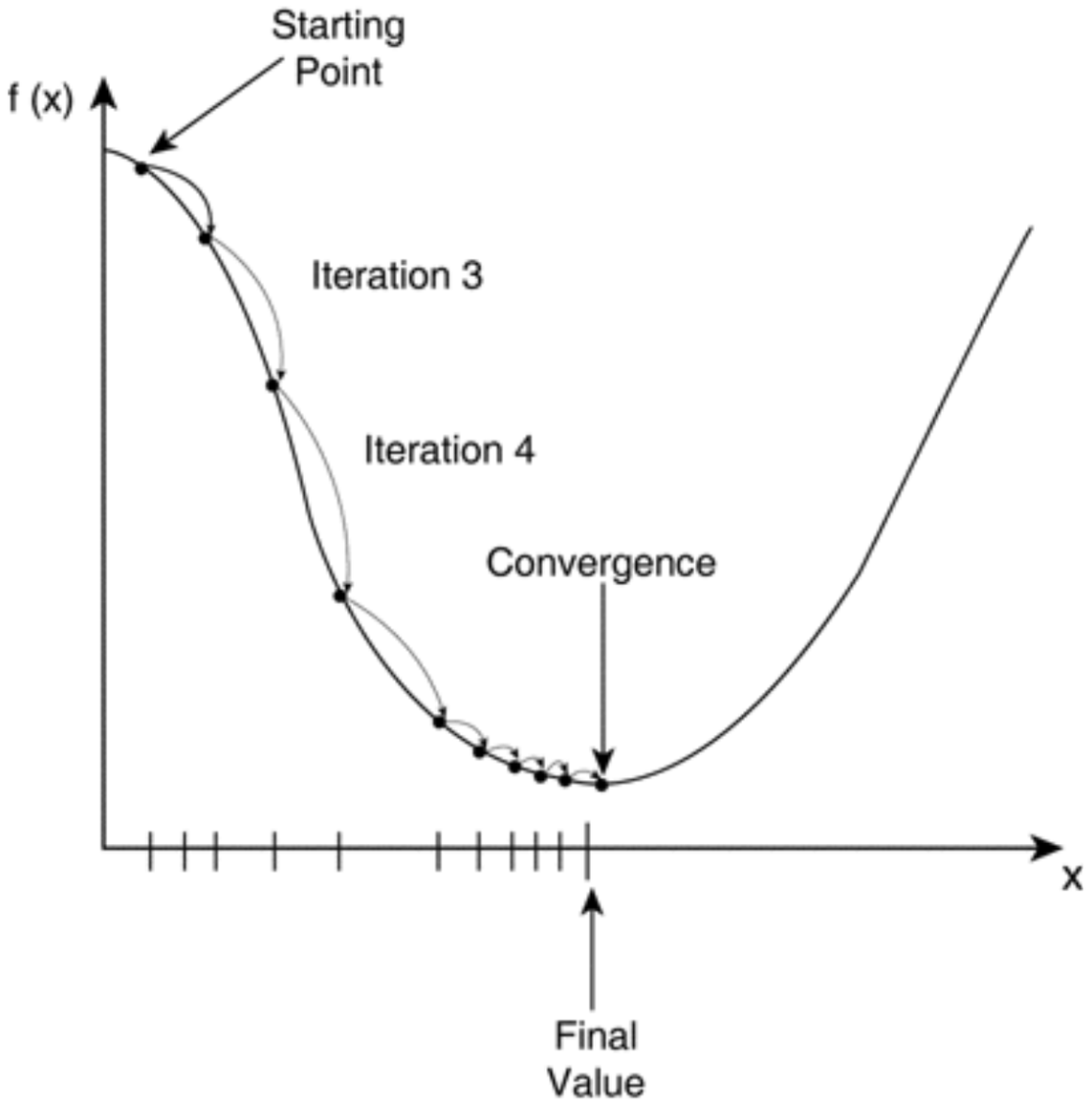
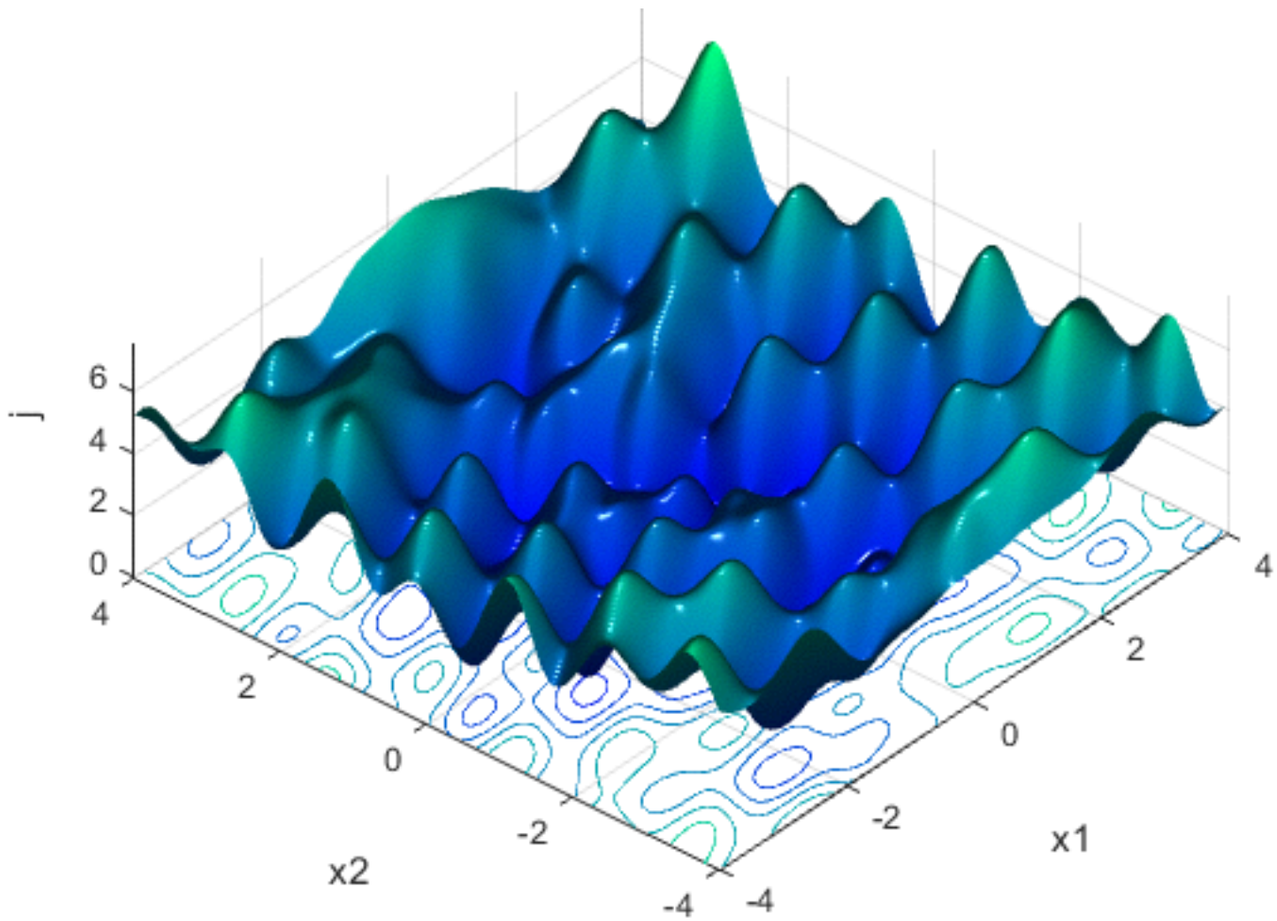
COMPUTER SCIENCE
INDIANA UNIVERSITY
School of Informatics and Computing
Bloomington

Optimization





Reminders/Comments

- Typos in notes: ongoing improvements, if you find any please tell me and I'll give you a gold star
  - e.g., last class had ln(2 pi d lambda) instead of ln(2 pi) - d ln(lambda)
  - I have uploaded a new version with a few typo fixes
- Appendix in notes has a discussion on second-order optimization

Thought question

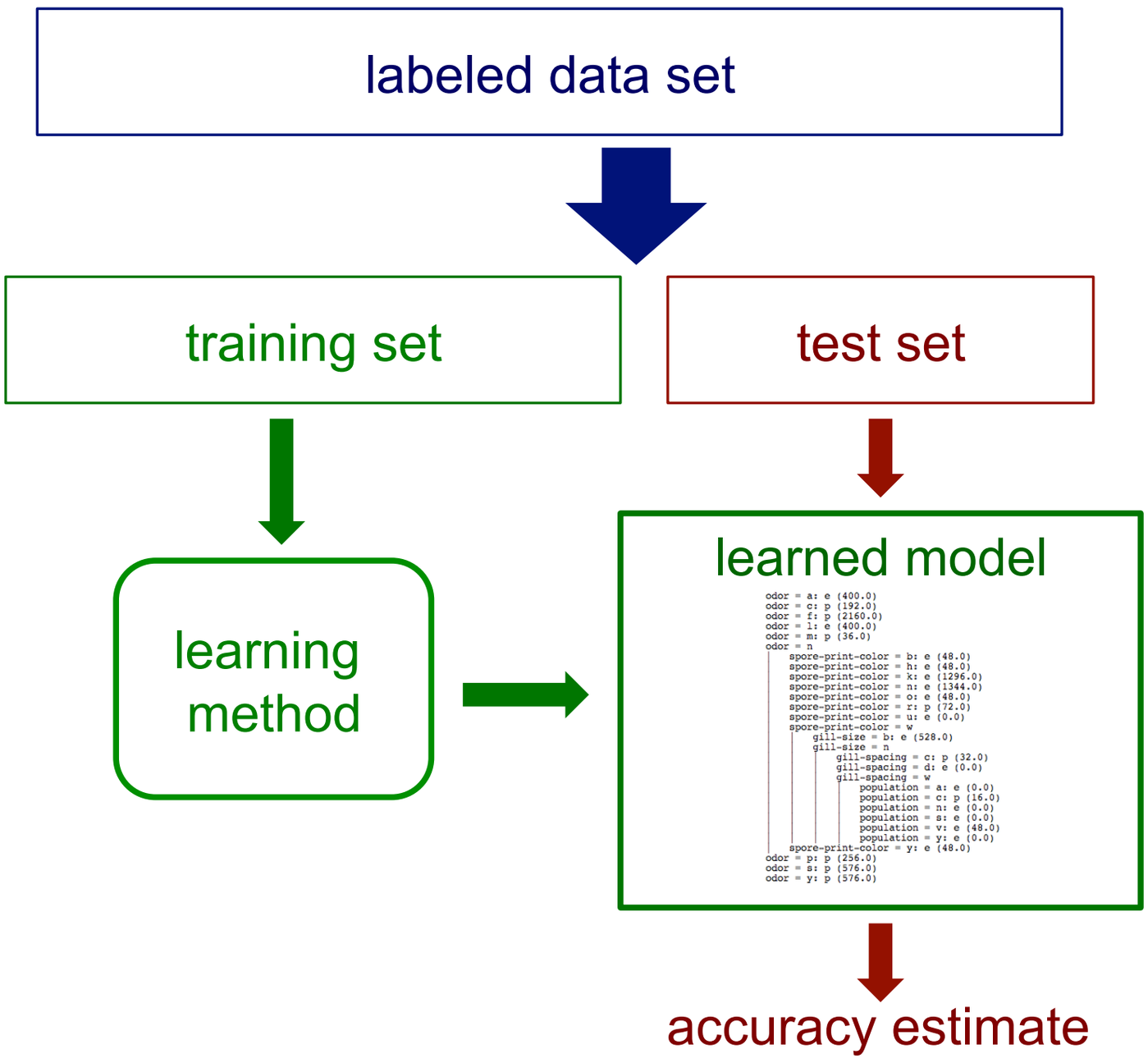
- “To model p(ylx), we need a functional form and a set of tunable coefficients. My question is how to determine which functional form to use? We may have one functional form and a set of coefficients give the same prediction as another functional form and another set of coefficients. How can we tell which model is the true model?”
  - Can always sanity check your assumptions, ensure your specification is sound; (e.g., did you appropriately convert categorical features)
  - From there, usually test the models empirically
  - Ultimate goal usually to get model that does best in terms of some performance metric (e.g., expected classification accuracy)
  - How do we tell how good our model will be, given only a (limited) batch of data?

Measuring error

- What if you train many different models on a batch of data, check their accuracy on that data, and pick the best one?

- What if the models you are testing are only different in terms of the regularization parameter lambda that they use? What will you find?

Simulating generalization error



Simulating generalization error

- Now we have one model, trained similarly to how it will be trained, and a measure of accuracy on new data (but distributed identically to trained data)

- What if we pick the model with the best test accuracy?

- Further, how do we generally do this for other metrics? What if we want a model that balances between two metrics? Or one that does reasonably well, but has low variance?

Thought question

- “After reading about how to calculate the posterior distribution, I wonder about the choice of prior distribution. According to the two examples and the context (Example 8 and 9) mentioned in the notes, when the size of the data set is large enough, both MAP and ML estimate converge to the same model. Here’s the things, the size of data collected today is increasing tremendously, is it make sense that we no longer need to use the prior distribution, or in other words, is it possible we won’t need to spend time on prior estimate? I think the answer is, it depends. For some disciplines such as zoology and archaeological geography, it is hard to obtain data, so priori knowledge and priori estimate is important. However, for gene science and big data science, maybe priori estimate will be less important in the future.”
  - For modern complex models, might still need priors/regularization
  - Some forms of regularizers are designed to be used even as data increases in size (e.g., l1 for feature selection and sparsity)

Bias-variance summary

$$\mathrm{MSE}(\mathbf{w}, \boldsymbol{\omega}) = \mathbb{E}[||\mathbf{w} - \boldsymbol{\omega}||_2^2]$$

$$= \mathbb{E}[\sum_{j=1}^{d} (\mathbf{w}_j - \boldsymbol{\omega}_j)^2]$$

$$= \sum_{j=1}^{d} \mathbb{E}[(\mathbf{w}_j - \boldsymbol{\omega}_j)^2]$$

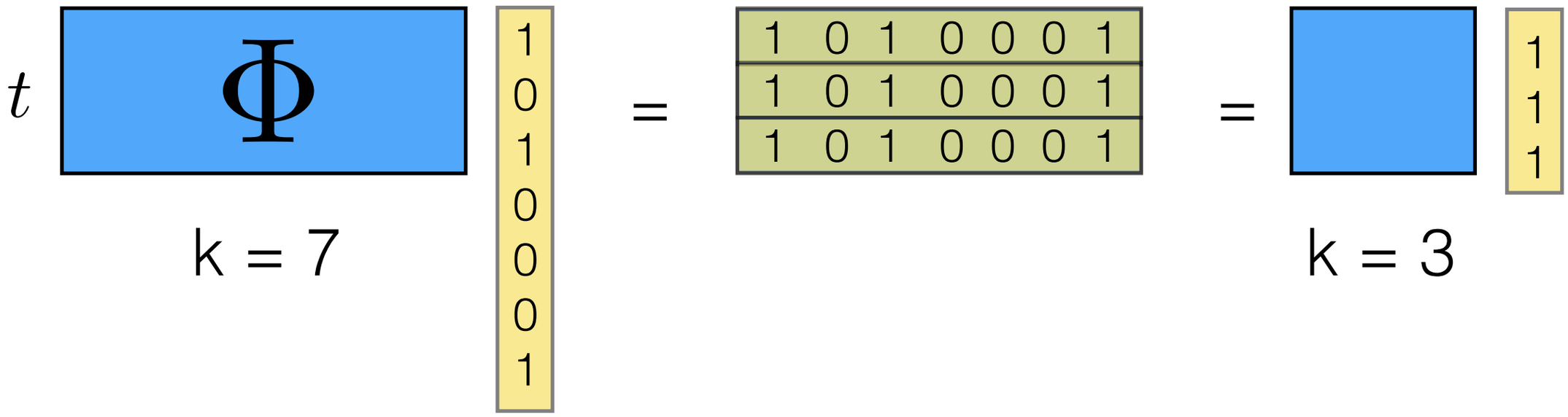
$$= \sum_{j=1}^{d} \mathrm{Bias}(\mathbf{w}_j, \boldsymbol{\omega}_j)^2 + \mathrm{Var}(\mathbf{w}_j)$$

$$\mathrm{Bias}(\mathbf{w}_j, \boldsymbol{\omega}_j) = \mathbb{E}[\mathbf{w}_j] - \boldsymbol{\omega}_j$$

l2 regularization trades off bias and variance

l1 regularization

- Feature selection, as well as preventing large weight



- How do we solve this optimization?

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_1$$

Why should you care about the solution strategies?

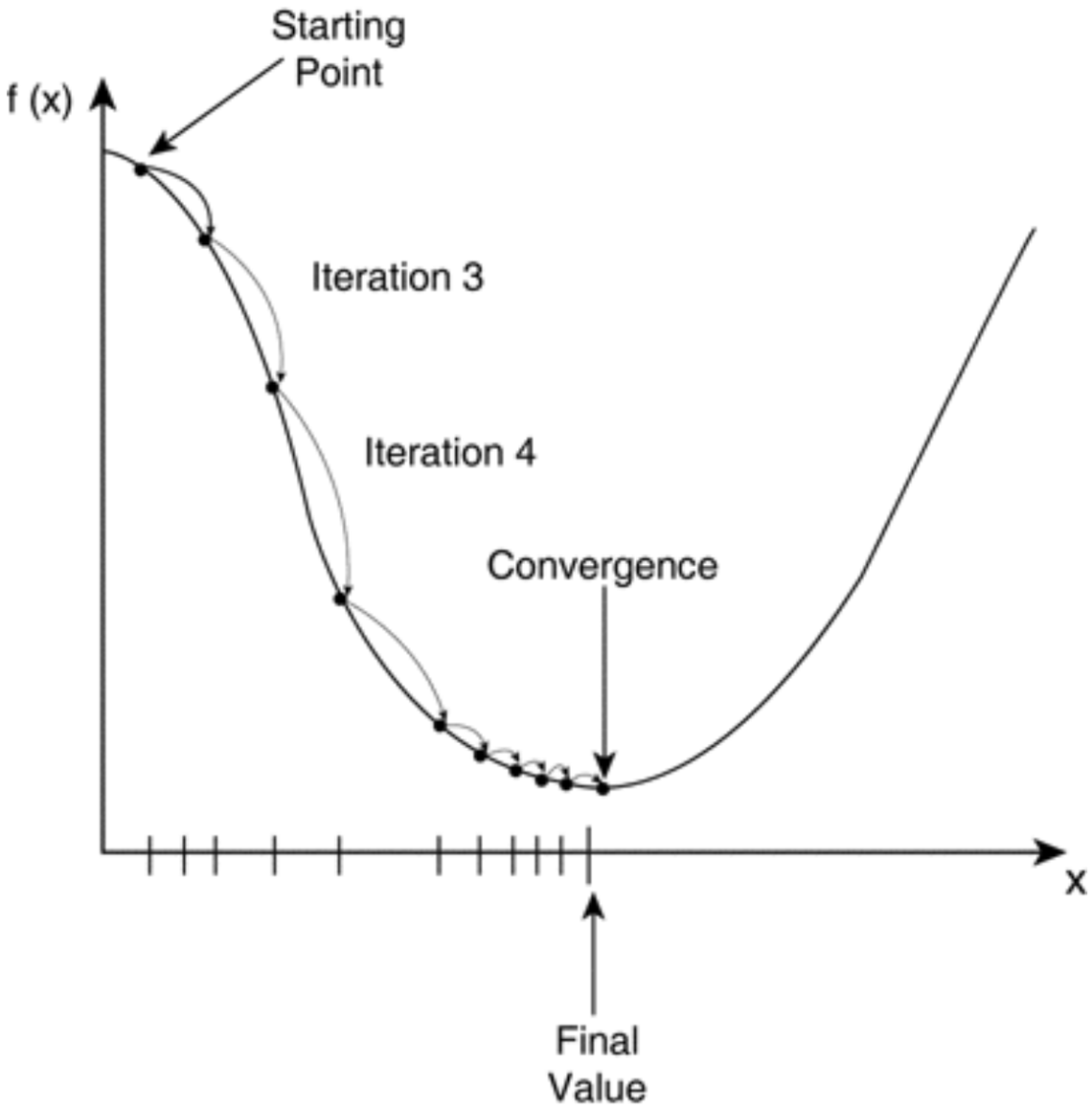
- Understanding the optimization approaches behind the algorithms makes you more effectively choose which algorithm to run

- Understanding the optimization approaches makes you formalize your problem more effectively
  - otherwise you might formalize a very hard optimization problem; sometimes with minor modifications, can significantly simplify for the solvers, without impacting properties of solution significantly

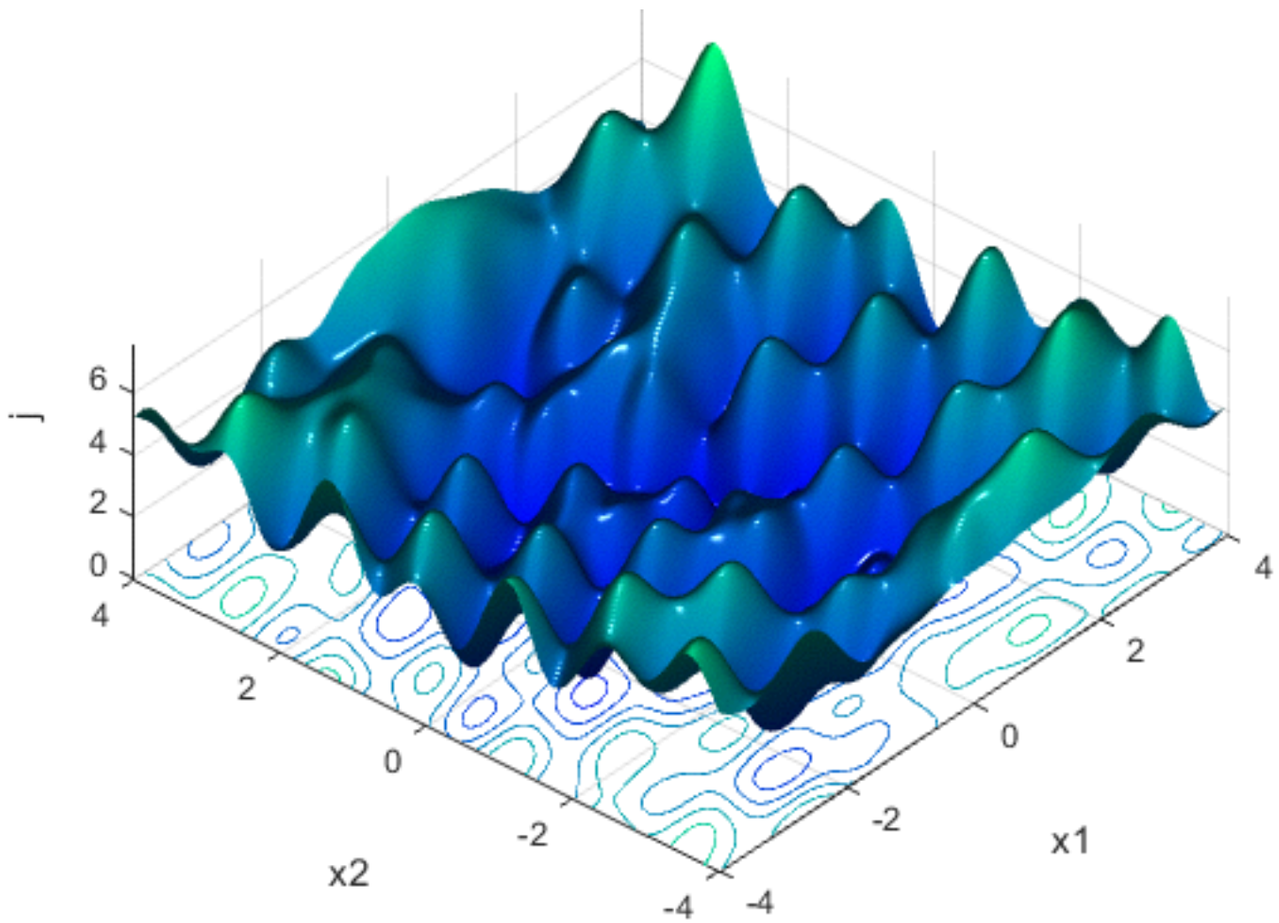
- When you want to do something outside the given packages or solvers (which is often true)

- …also its fun!

Gradient descent intuition



Convex function



Non-convex function

Why gradient descent?

- Taylor series expansion with
  - First order for gradient second
  - Second order for Newton-Raphson method (also called second-order gradient descent)

Taylor series expansion

A function $f(x)$ in the neighborhood of point x_0, can be approximated using the Taylor series as

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n,$$

where $f^{(n)}(x_0)$ is the n-th derivative of function $f(x)$ evaluated at point x_0.

e.g. $$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0).$$

Multivariate Taylor series

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \cdot (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \cdot H_{f(\mathbf{x}_0)} \cdot (\mathbf{x} - \mathbf{x}_0),$$

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_k} \right)$$

$$H_{f(\mathbf{x})} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_k} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & & \\ \vdots & & \ddots & \\ \frac{\partial^2 f}{\partial x_k \partial x_1} & & & \frac{\partial^2 f}{\partial x_k^2} \end{bmatrix}$$

First order

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \cdot (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \cdot H_{f(\mathbf{x}_0)} \cdot (\mathbf{x} - \mathbf{x}_0) ,$$

What if we only have access to the function and the first derivation (gradient) and not the Hessian?

Second-order optimization

- If have Hessian, can use second-order techniques
  - Mostly removes the need for a step-size parameter, and/or makes the choice of this parameter much less sensitive
- For certain situations, computing the Hessian is too expensive (in space and computation) and so first-order methods are used OR quasi-second order (LBFGS)
  - e.g., huge number of features

Second-order: Newton-Raphson for single-variate setting

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0).$$

$$f'(x) \approx f'(x_0) + (x - x_0)f''(x_0) = 0.$$

Solving this equation for x gives us

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)}.$$

Iterating gives us:

$$x^{(i+1)} = x^{(i)} - \frac{f'(x^{(i)})}{f''(x^{(i)})}.$$

Second-order: Newton-Raphson for multi-variate setting

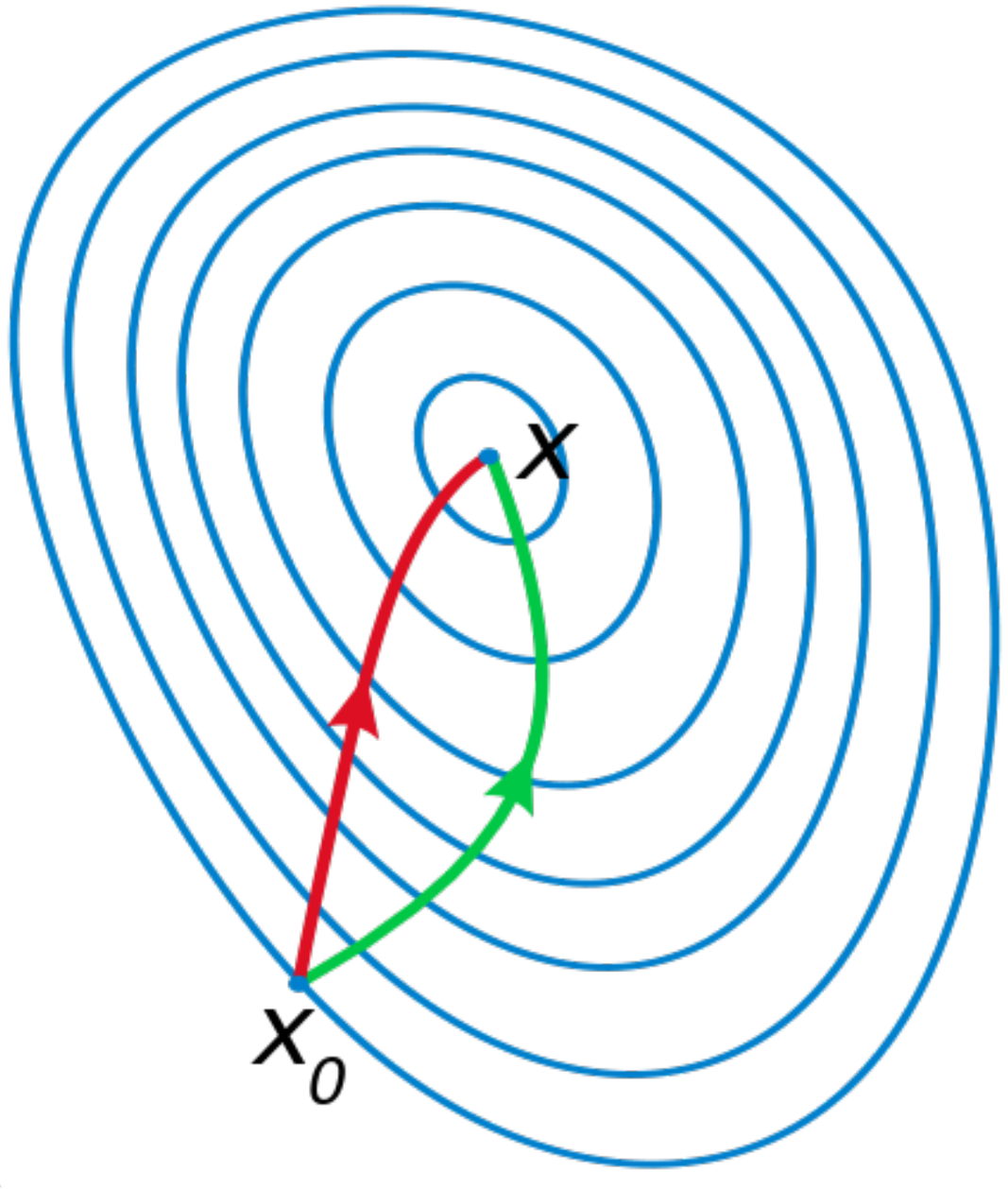
$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \cdot (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \cdot H_{f(\mathbf{x}_0)} \cdot (\mathbf{x} - \mathbf{x}_0) ,$$

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \left(H_{f(\mathbf{x}^{(i)})}\right)^{-1} \cdot \nabla f(\mathbf{x}^{(i)}),$$

Intuition for first and second order

- Locally approximate function at current point
- For first order, locally approximate as linear and step in the direction of the minimum of that linear function
- For second order, locally approximate as quadratic and step in the direction of the minimum of that quadratic function
  - a quadratic approximation is more accurate
- What happens if the true function is quadratic?

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \left(H_{f(\mathbf{x}^{(i)})}\right)^{-1} \cdot \nabla f(\mathbf{x}^{(i)}),$$



Newton in red

Gradient descent

Algorithm 1: Batch Gradient Descent(Err, $\mathbf{X}, \mathbf{y}$)

1: // A non-optimized, basic implementation of batch gradient descent
2: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
3: err $\leftarrow \infty$
4: tolerance $\leftarrow 10e^{-4}$
5: $\alpha \leftarrow 0.1$
6: **while** $|\text{Err}(\mathbf{w}) - \text{err}| > \text{tolerance}$ **do**
7: err $\leftarrow \text{Err}(\mathbf{w})$
8: // The step-size α should be chosen by line-search
9: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \text{Err}(\mathbf{w}) = \mathbf{w} - \alpha \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$
10: **return** $\mathbf{w}$

Recall: for error function $E(\mathbf{w})$ goal is to solve $\nabla E(\mathbf{w}) = \mathbf{0}$

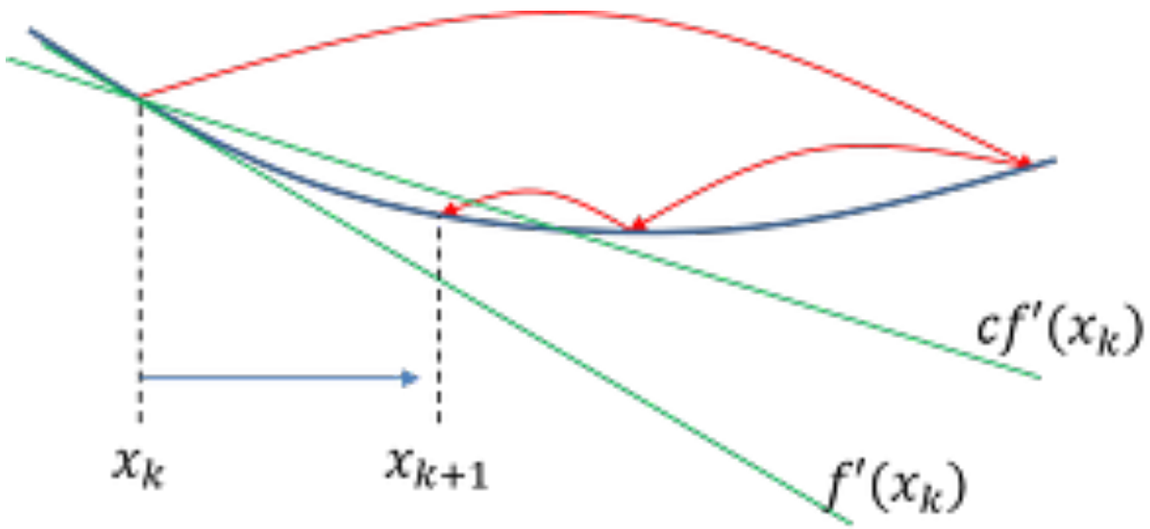
Line search

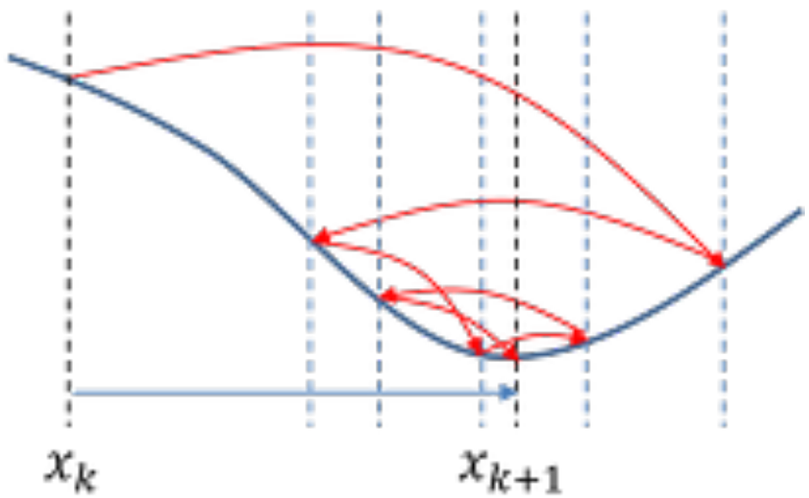
Want step-size such that

$$\alpha = \arg\min_{\alpha} E(\mathbf{w} - \alpha \nabla E(\mathbf{w}))$$

Backtracking line search:

1. Start with relatively large α (say $\alpha = 1$)
2. Check if $E(\mathbf{w} - \alpha \nabla E(\mathbf{w})) < E(\mathbf{w})$
3. If yes, use that α
4. Otherwise, decrease α (e.g., $\alpha = \alpha/2$), and check again





What is the second-order gradient descent update?

Algorithm 1: Batch Gradient Descent(Err, $\mathbf{X}, \mathbf{y}$)

// A non-optimized, basic implementation of batch gradient descent
$\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
err $\leftarrow \infty$
tolerance $\leftarrow 10e^{-4}$
$\alpha \leftarrow 0.1$
while $|\text{Err}(\mathbf{w}) - \text{err}| > \text{tolerance}$ **do**
    err $\leftarrow \text{Err}(\mathbf{w})$
    // The step-size α should be chosen by line-search
    $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \text{Err}(\mathbf{w}) = \mathbf{w} - \alpha \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$
return $\mathbf{w}$

Batch optimization

- When might we want to use second-order gradient descent?

- What are some issues with batch gradient descent?

Stochastic gradient descent

Algorithm 2: Stochastic Gradient Descent$(E, \mathbf{X}, \mathbf{y})$

1: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
2: **for** $t = 1, \ldots, n$ **do**
3: // For some settings, we need the step-size α_t to decrease with time
4: $\mathbf{w} \leftarrow \mathbf{w} - \alpha_t \nabla E_t(\mathbf{w}) = \mathbf{w} - \alpha_t(\mathbf{x}_t^\top \mathbf{w} - y_t)\mathbf{x}_t$
5: **end for**
6: **return** $\mathbf{w}$

For batch error: $\hat{E}(\mathbf{w}) = \sum_{t=1}^{n} E_t(\mathbf{w})$
e.g., $E_t(\mathbf{w}) = (\mathbf{x}_t^\top \mathbf{w} - y_t)^2$
$\hat{E}(\mathbf{w}) = \sum_{t=1}^{n} E_t(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$
$\nabla \hat{E}(\mathbf{w}) = \sum_{t=1}^{n} \nabla E_t(\mathbf{w})$
$E(\mathbf{w}) = \int_{\mathcal{X}} \int_{\mathcal{Y}} f(\mathbf{x}, y)(\mathbf{x}^\top \mathbf{w} - y)^2 dy d\mathbf{x}$

- Stochastic gradient descent (stochastic approximation) minimizes with an unbiased sample of the gradient $\mathbb{E}[\nabla E_t(\mathbf{w})] = \nabla E(\mathbf{w})$

Stochastic gradient descent

$$\mathbb{E}\left[\frac{1}{n}\nabla\hat{E}(\mathbf{w})\right] = \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^{n}\nabla E_i(\mathbf{w})\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[\nabla E_i(\mathbf{w})]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[\nabla E(\mathbf{w})]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\nabla E(\mathbf{w})$$

$$= \nabla E(\mathbf{w})$$

Stochastic gradient descent

- Can also approximate gradient with more than one sample (e.g., mini-batch), as long as $\mathbb{E}[\nabla E_t(\mathbf{w})] = \nabla E(\mathbf{w})$

- Proof of convergence and conditions on step-size: Robbins-Monro ("A Stochastic Approximation Method", Robbins and Monro, 1951)

- A big focus in recent years in the machine learning community; many new approaches for improving convergence rate, reducing variance, etc.

Whiteboard

- **Exercise**: derive an algorithm to compute the solution to l1-regularized linear regression (i.e., MAP estimation with a Gaussian likelihood p(y l x, w) and Laplace prior)
  - First write down the Laplacian
  - Then write down the MAP optimization
  - Then determine how to solve this optimization
- Generalized linear models