Lecture 7 – Classification

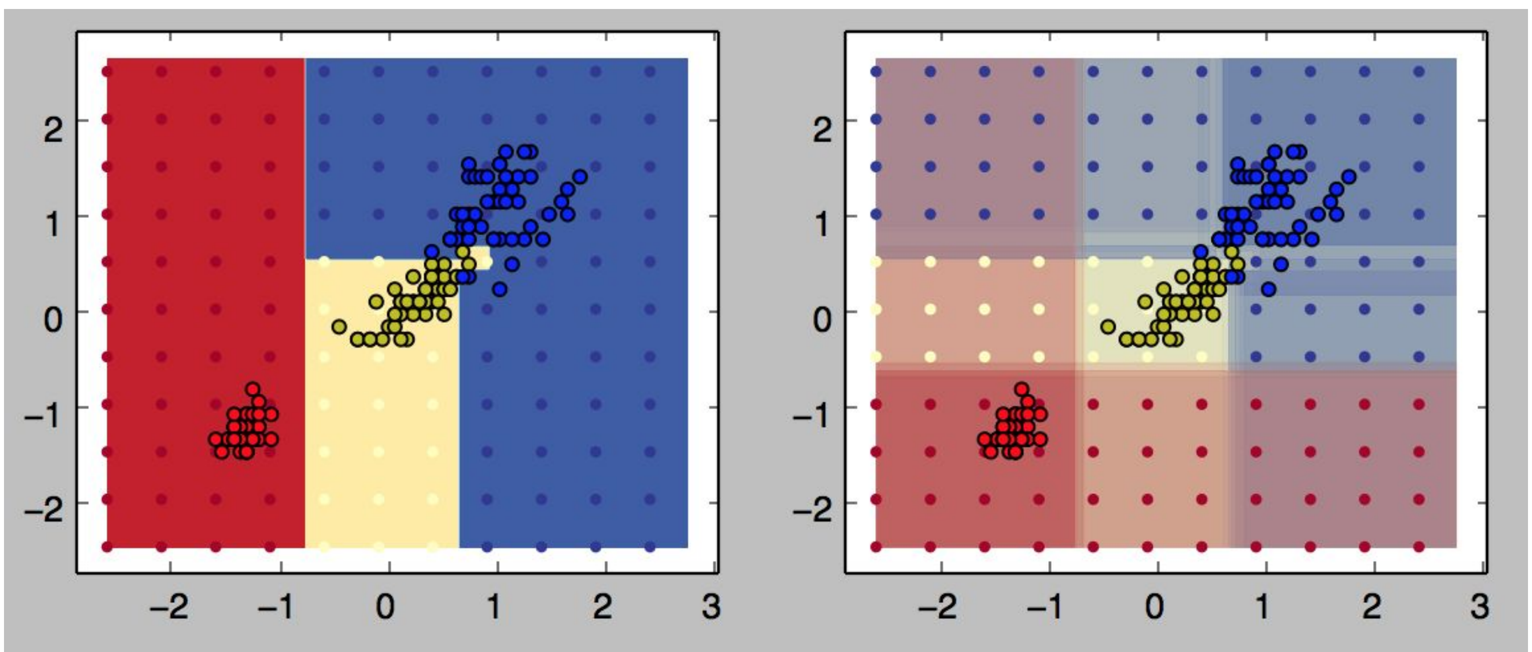
Machine Learning

Queens College

Classification

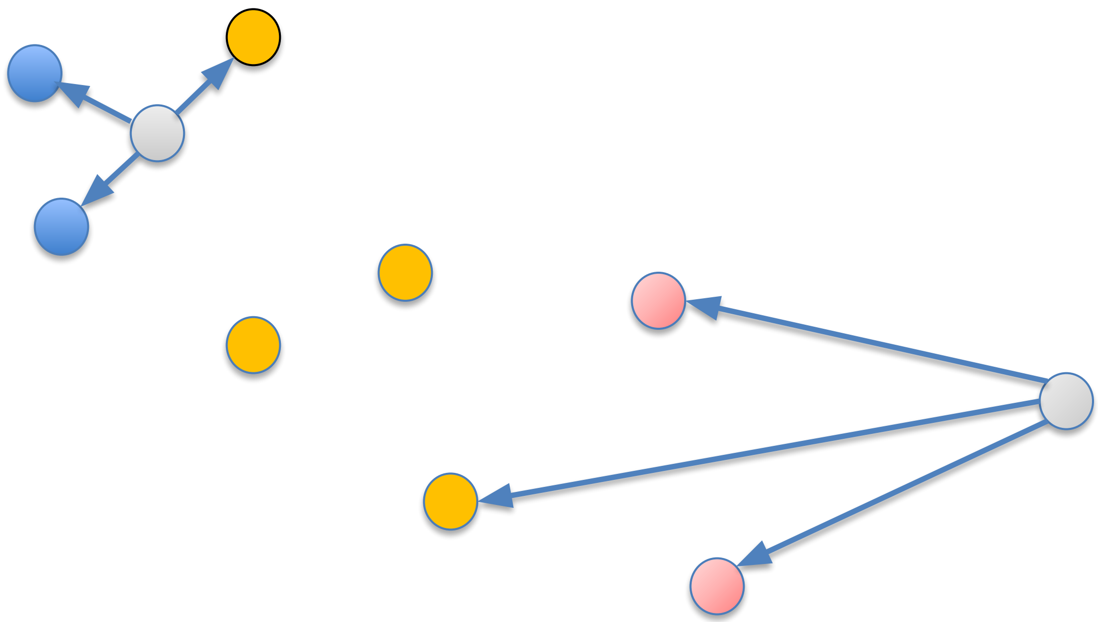
- Identify which of *c* classes a data point, **x**, belongs to.
- **x** is a column vector of features (D dimensional).

Decision Tree **Random Forest**



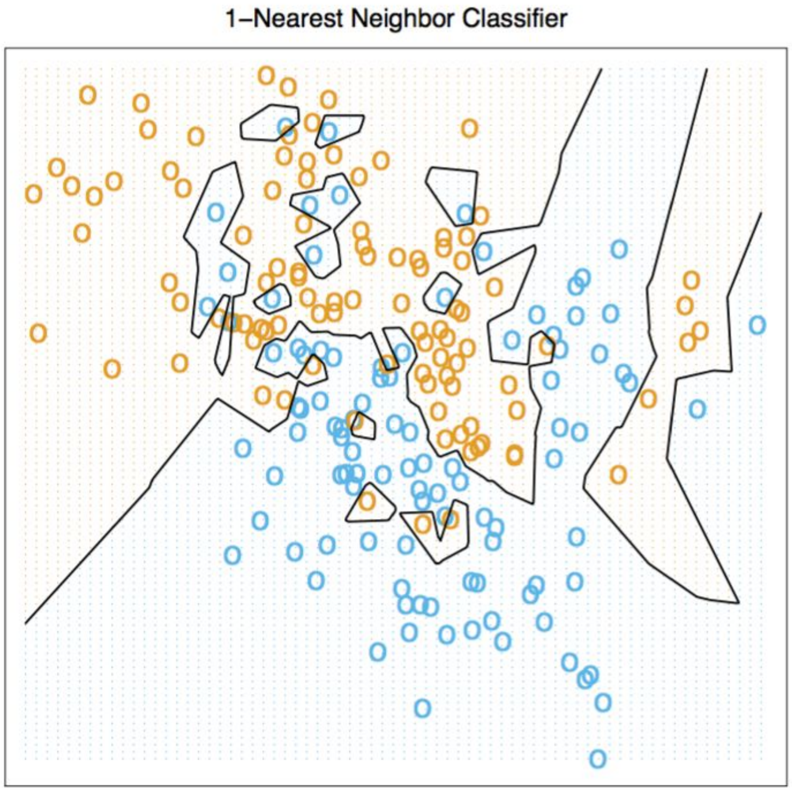
k-nearest neighbors

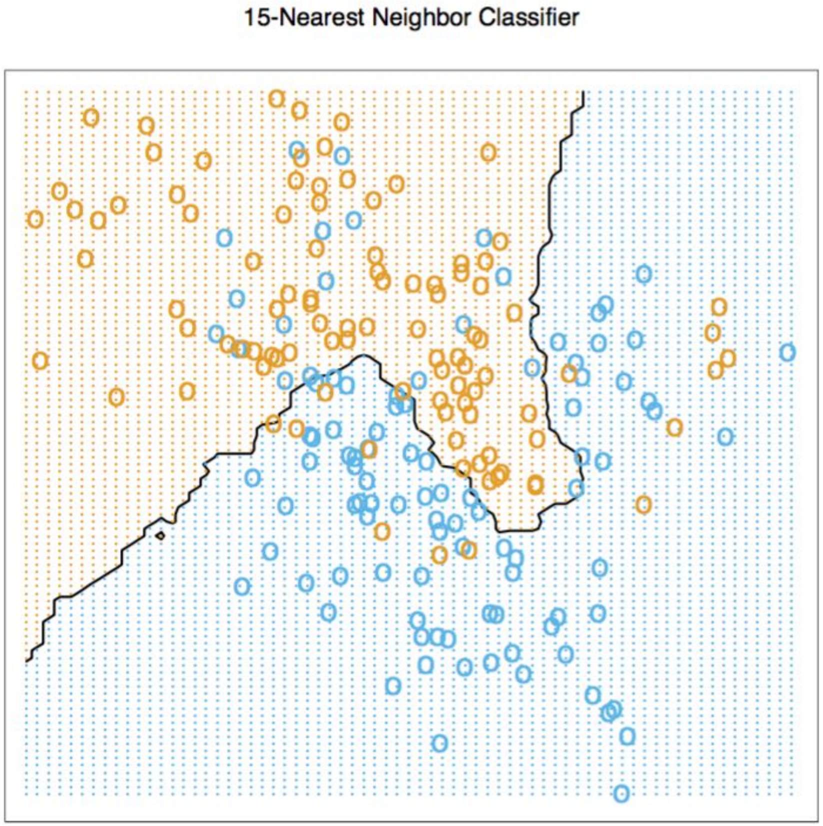
- For each test data, find its k nearest neighbors in the training data.
- The label of the test data is the majority of its neighbors.
- Example: k = 3



k-nearest neighbors

- K is critical
- no probabilistic foundation, practically useful
  - Example (plot_classification_knn.py), diff k? uniform vs distance based weight.
  - Plot_classification_knn_2.py
- Bias-variance decomposition (future)





Probability-based approach

- Use posterior probability: p(Y|X)

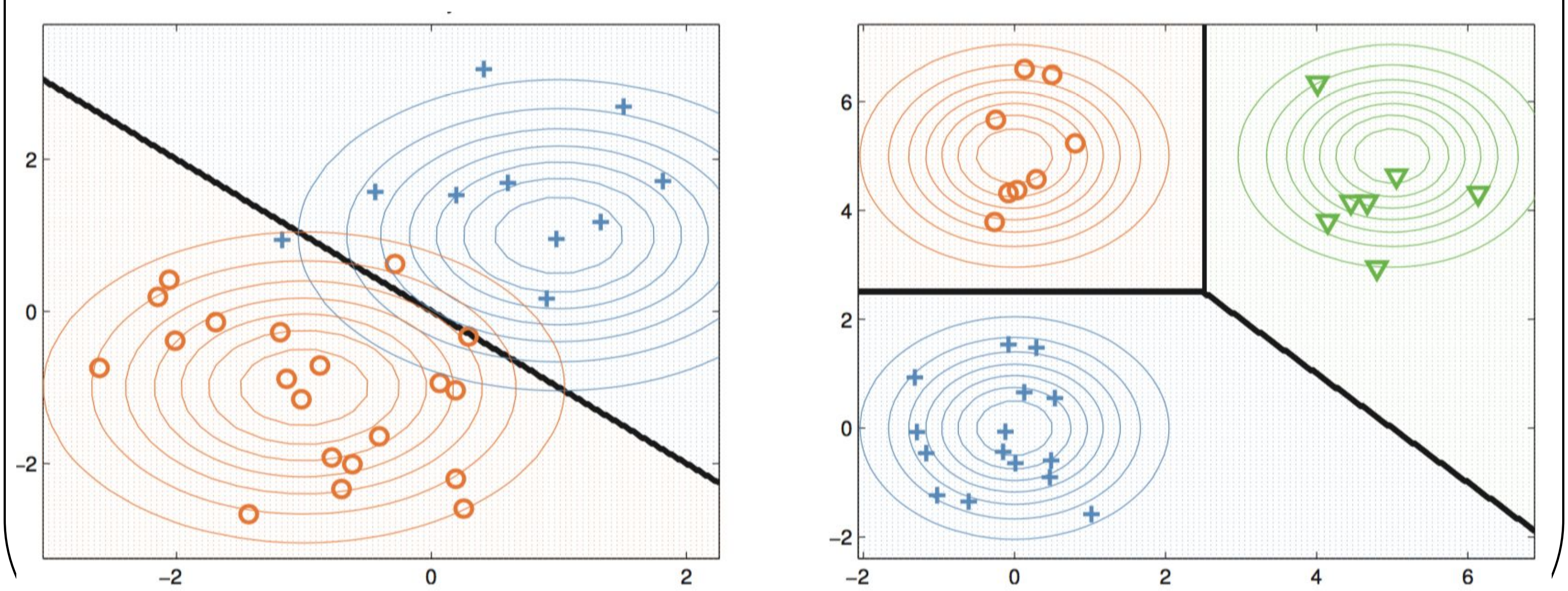
$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

- p(x) = sum of all
- p(c|x) is proportional to the numerator
- c* = argmax p(c|x)
- Naïve Bayesian assumption

$$p(x_1, x_2, \ldots, x_n|c) = p(x_1|c)p(x_2|c)\cdots p(x_n|c)$$

Linear Discriminant Analysis (LDA)

- c* = argmax p(c|x)
- Boundaries between regions are **linear**



Linear Discriminant Analysis (LDA)

- Not Latent Dirichlet Allocation (also LDA)

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

- Assumption 1: p(x|k) is a Gaussian distribution

$$\frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \boldsymbol{\Sigma}_k^{-1}(x-\mu_k)}$$

- Assumption 2: the covariance matrix for all classes are the same

$$\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma} \ \forall k$$

Linear Discriminant Analysis (LDA)

- p(x|k) = $f_k(x)$

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

- p(k) = π_k $\sum_{k=1}^{K} \pi_k = 1$

$$\Pr(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^{K} f_\ell(x)\pi_\ell}.$$

- p(k|x) > p(l | x) iff log(p(k|x) / p(l | x)) > 0
- Avoid the denominator
- Calculation

Linear Discriminant Analysis (LDA)

$$\log \frac{\Pr(G=k|X=x)}{\Pr(G=\ell|X=x)} = \log \frac{f_k(x)}{f_\ell(x)} + \log \frac{\pi_k}{\pi_\ell}$$

$$= \log \frac{\pi_k}{\pi_\ell} - \frac{1}{2}(\mu_k + \mu_\ell)^T \mathbf{\Sigma}^{-1}(\mu_k - \mu_\ell)$$

$$+ x^T \mathbf{\Sigma}^{-1}(\mu_k - \mu_\ell),$$

- Linear discriminant function

$$\delta_k(x) = x^T \mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k + \log \pi_k$$

$$G(x) = \operatorname{argmax}_k \delta_k(x).$$

Linear Discriminant Analysis (LDA)

- Linear discriminant function

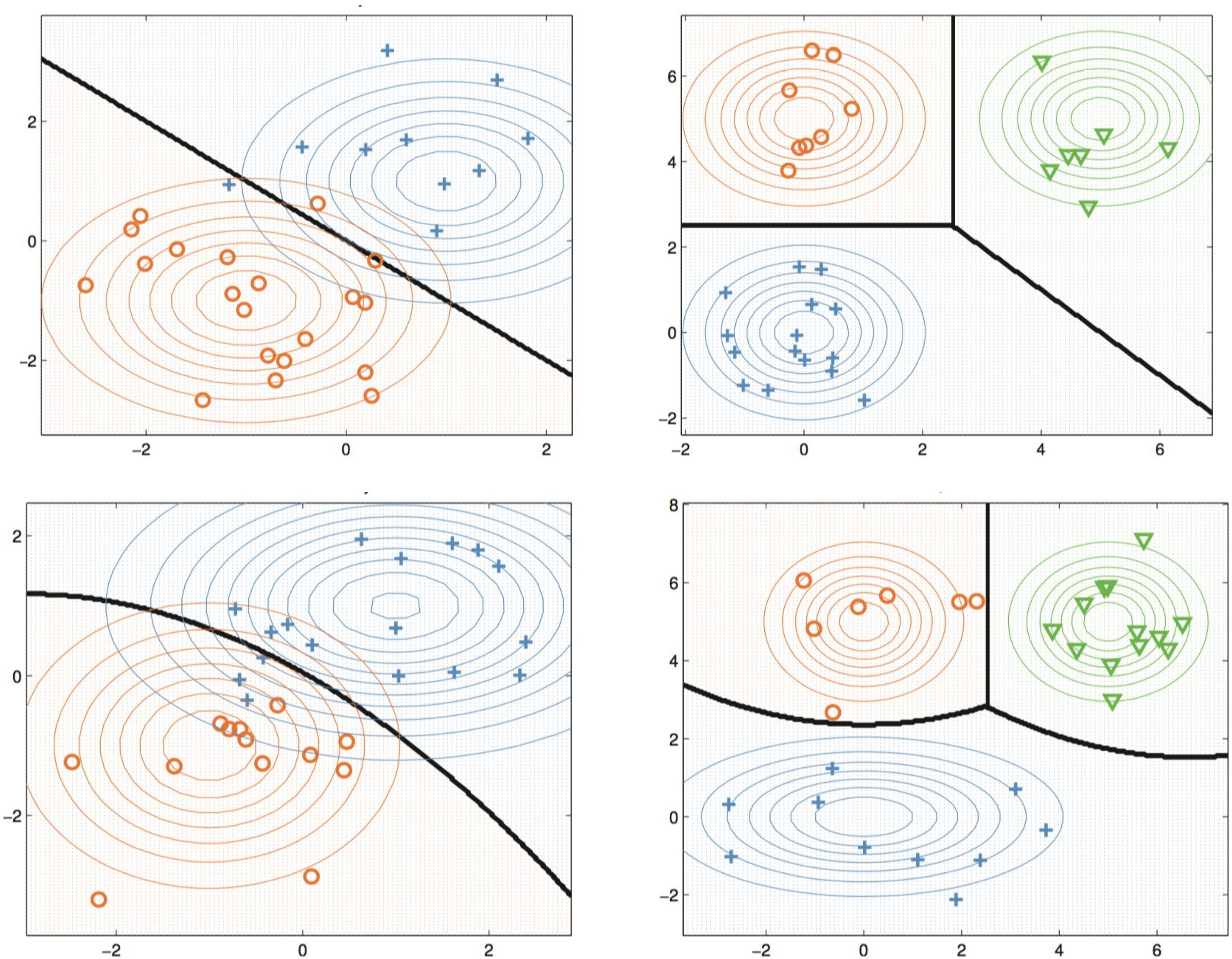
$$\delta_k(x) = x^T \mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k + \log \pi_k$$

$$G(x) = \text{argmax}_k \delta_k(x).$$

- prior, sample mean,
- pooled sample covariance matrix
- Why? MLE! How?
  - $\prod_{n=1}^{N} p(x_i, y_i)$
  - Prior, mean, cov can be derived separately

- $\hat{\pi}_k = N_k/N$, where N_k is the number of class-k observations;
- $\hat{\mu}_k = \sum_{g_i=k} x_i/N_k$;
- $\hat{\mathbf{\Sigma}} = \sum_{k=1}^{K} \sum_{g_i=k}(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T/(N-K)$.

From linear to quadratic (Asm 2)



Quadratic Discriminant Analysis (QDA)

- Quadratic discriminant function

$$\delta_k(x) = -\frac{1}{2}\log|\mathbf{\Sigma}_k| - \frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}_k^{-1}(x-\mu_k) + \log\pi_k.$$

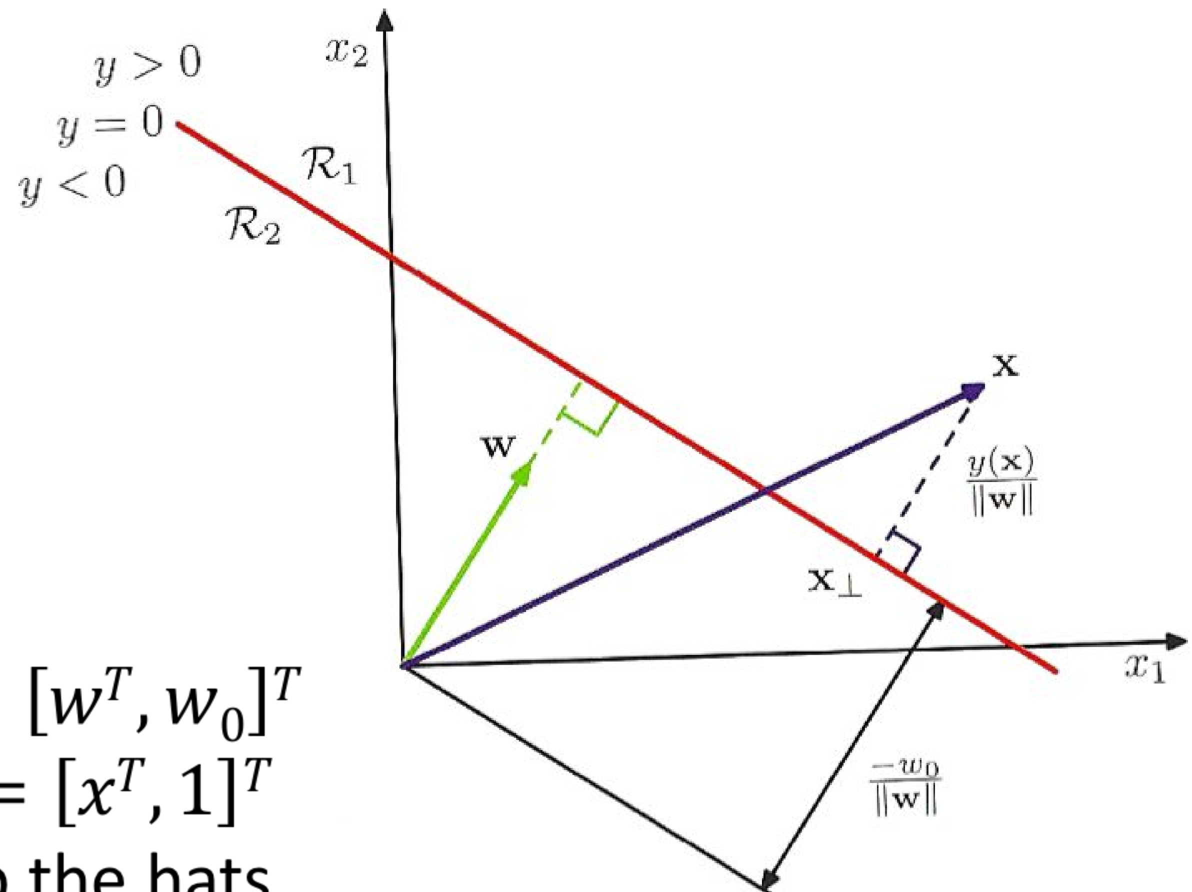
$$G(x) = \text{argmax}_k \delta_k(x).$$

- prior, sample mean,
- sample covariance matrix for each k, $\mathbf{\Sigma}_k$
- Examples: LDA/QDA, The function

Sample Variance/Covariance

- Variance: E[(x - μ)^2] = $\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$
- Issue: μ is the true (population) mean, unknown
- We use sample mean $\hat{\mu} = \frac{1}{\mathrm{N}} \sum x_i$
- $\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{\mu})^2$ biased as $\hat{\mu}$ comes from samples
- To compensate for this, divide by the degree of freedom, instead of the sample size (Bessel's correction)
  - DoF: sample size - # of parameters estimated
  - N – 1 (1 counts for the sample mean)
  - Only when $\hat{\mu}$ is estimated
  - When N is large, not much difference
- https://en.wikipedia.org/wiki/Bessel%27s_correction

Treating Classification as a Linear model



$y = w^T x + w_0$

Extend to $\widehat{w} = [w^T, w_0]^T$

$\hat{x} = [x^T, 1]^T$

$y = \widehat{w}^T \hat{x}$, Drop the hats

When $\|w\|$= 1, y = projection on w, with offset $-w_0$

Perceptron

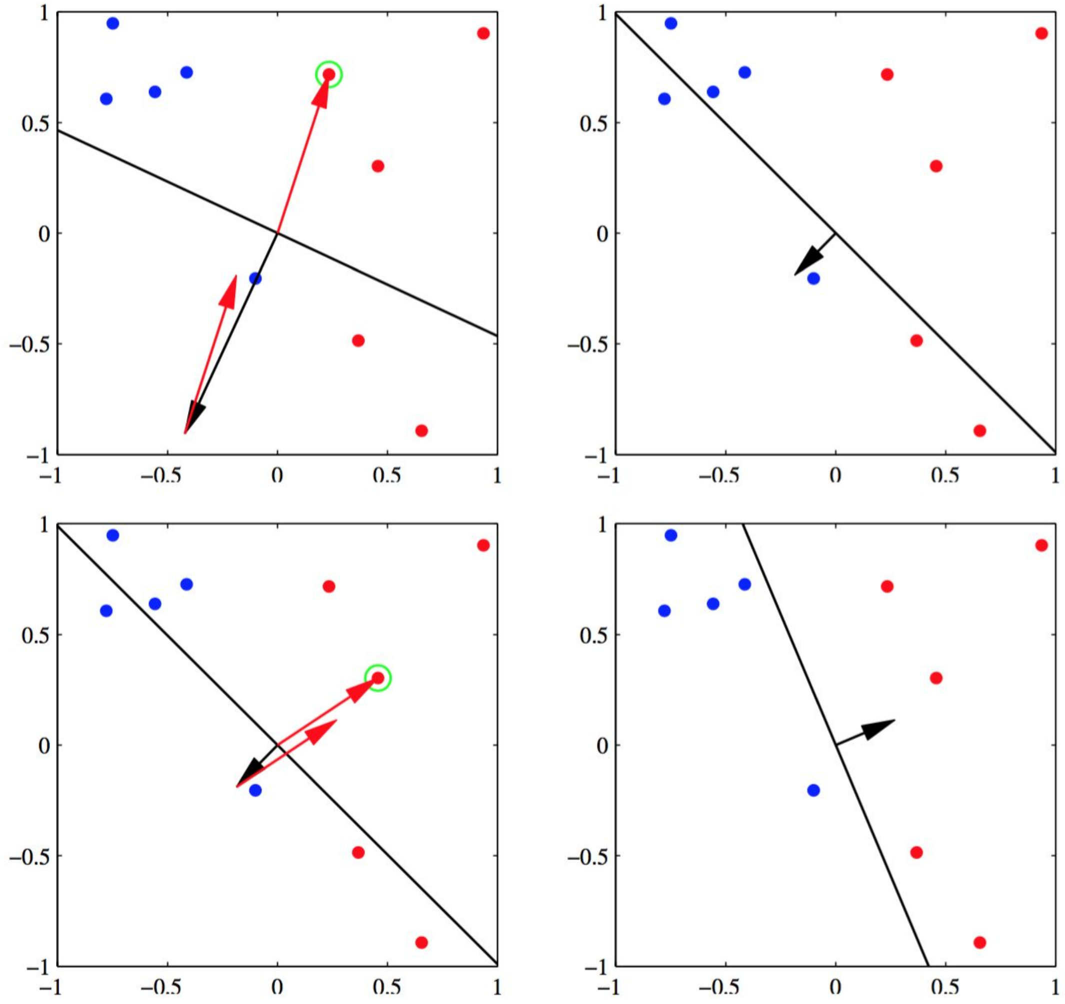
- Binary classification y = 1 or -1
  - $y(x) = f(w^T x)$
  - f: sign,   view along the normal direction (w)
  - 1D, 2D examples, penalty for misclassification?
- Learning:
  - error – when $\|w\|= 1$, means absolute values of distance of mislabeled points from the hyperplane.

$$\text{Error(w)} = -\sum_{n \in \mathcal{M}} t_n w^T x_n$$

- $w^{\tau+1} = w^{\tau} - \eta \nabla Error(w) = w^{\tau} + \eta \sum_{n \in \mathcal{M}} t_n x_n$
- η step size / learning rate
- Stochastic gradient descent
  - Pick one misclassified sample, update weights
  - $w^{\tau+1} = w^{\tau} + \eta' t_n x_n$

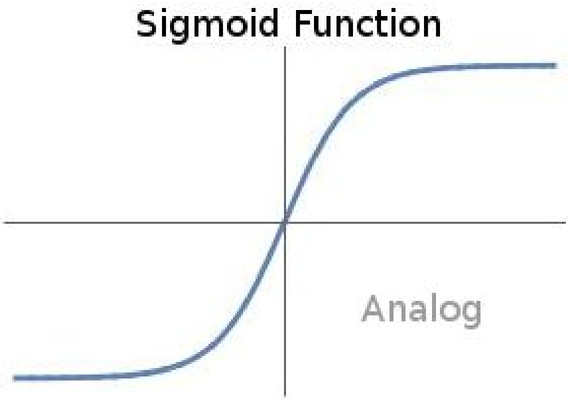
Learning Perceptron

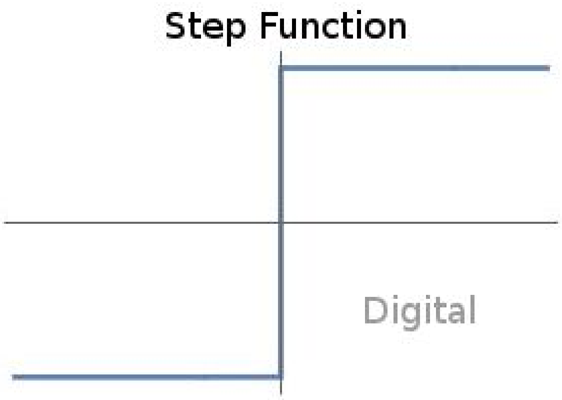
- Example: assuming $w_0 = 0$
- Left to right, top to bottom (r = +1, b = -1)



Learning Perceptron

- Issue:
  - Different initializations may go to different solutions
  - When existing separating plane, learning converges
  - Otherwise, will NOT converge
- Fix:
  - Make f(x) a softer function (differentiable)?
  - Sigmoid





Logistic Regression

- Logistic Regression
  - Linear Model for Classification
  - Name is misleading: not for regression!
  - Solid probabilistic interpretation
  - Learning converges

Odds-ratio

- Rather than thresholding, we'll relate the regression to the **class-conditional** probability.
- odds of success (y = 1) v.s. fail (y = 0)
  - If p(y=1|x) = 0.8 and p(y=0|x) = 0.2
  - Odds = 0.8/0.2 = 4
- Use a linear model to predict **odds** rather than a class label.

$$\frac{p(y=1|\vec{x})}{p(y=0|\vec{x})} = \vec{w}^T\vec{x}$$

- Label {0, 1} (in perceptron {-1, 1})

Logit – Log odds ratio function

$$\frac{p(y=1|\vec{x})}{p(y=0|\vec{x})} = \vec{w}^T\vec{x}$$

- LHS: 0 to infinity
- RHS: -infinity to infinity
- Use a log function.
  - Has the added bonus of dissolving the division leading to easy manipulation

$$\log \frac{p(y=1|\vec{x})}{p(y=0|\vec{x})}$$

$$\log \frac{p(y=1|\vec{x})}{1-p(y=1|\vec{x})}$$

$$logit(p(x)) = \log \frac{p(x)}{1-p(x)}$$

Logistic Regression

- A **linear model** used to predict log-odds ratio of two classes

$$\log \frac{p(y=1|\vec{x})}{1-p(y=1|\vec{x})} = \vec{w}^T \vec{x}$$

- Take exp on both sides, normalize

Previously (LDA)

- Assume the data is **generated** from a Gaussian distribution for each class (same cov).

$$p(\vec{x}|C_1) = N(\vec{x}|\vec{\mu_1}, \Sigma)$$

- Leads to linear log-odds

$$\begin{aligned}\log \frac{\Pr(G=k|X=x)}{\Pr(G=\ell|X=x)} &= \log \frac{f_k(x)}{f_\ell(x)} + \log \frac{\pi_k}{\pi_\ell} \\ &= \log \frac{\pi_k}{\pi_\ell} - \frac{1}{2}(\mu_k + \mu_\ell)^T \mathbf{\Sigma}^{-1}(\mu_k - \mu_\ell) \\ &\quad + x^T \mathbf{\Sigma}^{-1}(\mu_k - \mu_\ell),\end{aligned}$$

Now: logistic regression

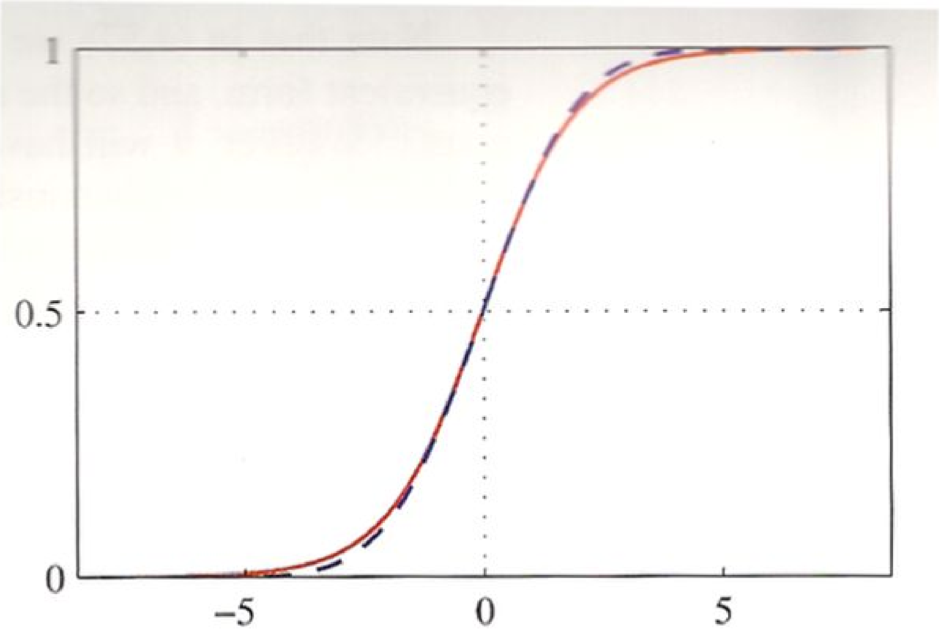
- Still linear model for the log-odd

$$\log \frac{p(y=1|\vec{x})}{1-p(y=1|\vec{x})} = \vec{w}^T \vec{x}$$

- But no Gaussian assumptions on p(x|c)
- Instead, find the posterior p(y|x) directly

Sigmoid function

$$\sigma(x) = \frac{1}{1 + exp(-x)}$$



- Squashing function to map the reals to a finite domain.

$$\sigma : \mathbb{R} \to (0, 1)$$

Maximum likelihood learning

- Still linear model for the log-odd

$$\log \frac{p(y=1|\vec{x})}{1-p(y=1|\vec{x})} = \vec{w}^T \vec{x}$$

- But no Gaussian assumptions on p(x|c)
- Instead, find the posterior p(y|x) directly
- The likelihood of the data

$$\prod_{i=1}^{N} p(y = t_i \mid x_i, \theta\ )$$

Model and Likelihood

- Model: $p(\mathcal{C}_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}\right)$

- Likelihood: $p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} \left\{1 - y_n\right\}^{1-t_n}$

- Cross-entropy loss (negative-log-likelihood):

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N} \left\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\right\}$$

Gradients

- Gradient w.r.t. w

$$\frac{d\sigma}{da} = \sigma(1 - \sigma).$$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n$$

- Updating: $w^{\tau+1} = w^{\tau} - \eta \nabla Error(w)$
- Compare with perceptron gradient

$$\nabla Error(w) = \sum_{n \in \mathcal{M}} (-t_n) x_n$$

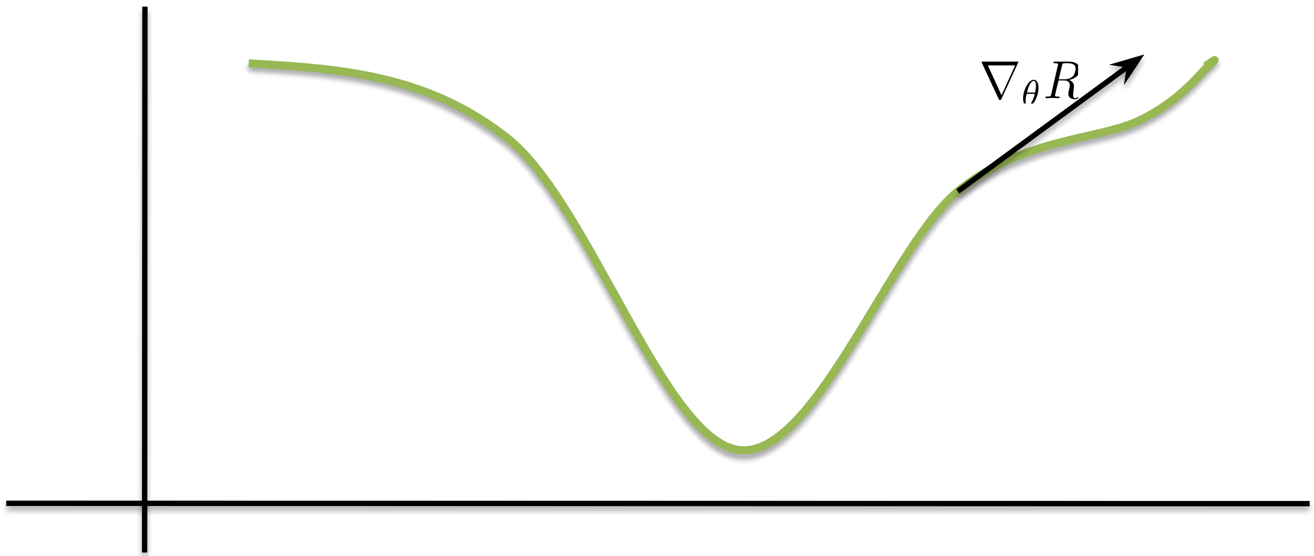
Gradient Descent

- Take a guess.
- Move in the direction of the negative gradient
- Jump again.

$$w_{n+1} = w_n - \eta \nabla_{\vec{w}} E(w_n)$$

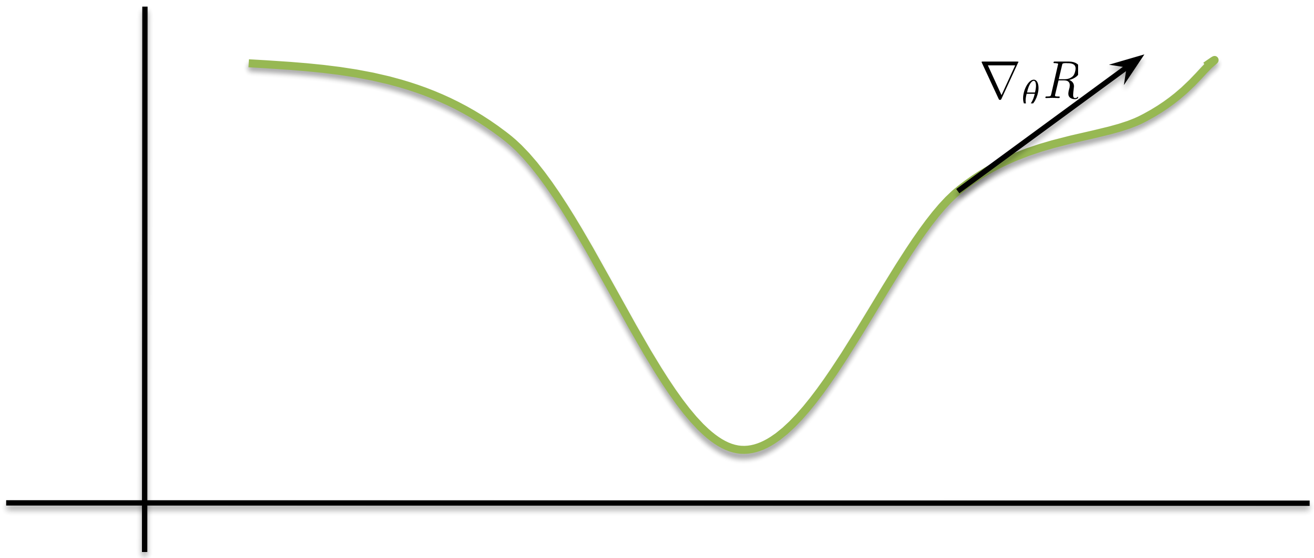
Gradient Descent

- The Gradient is defined (though we can’t solve directly)
- Points in the direction of fastest increase



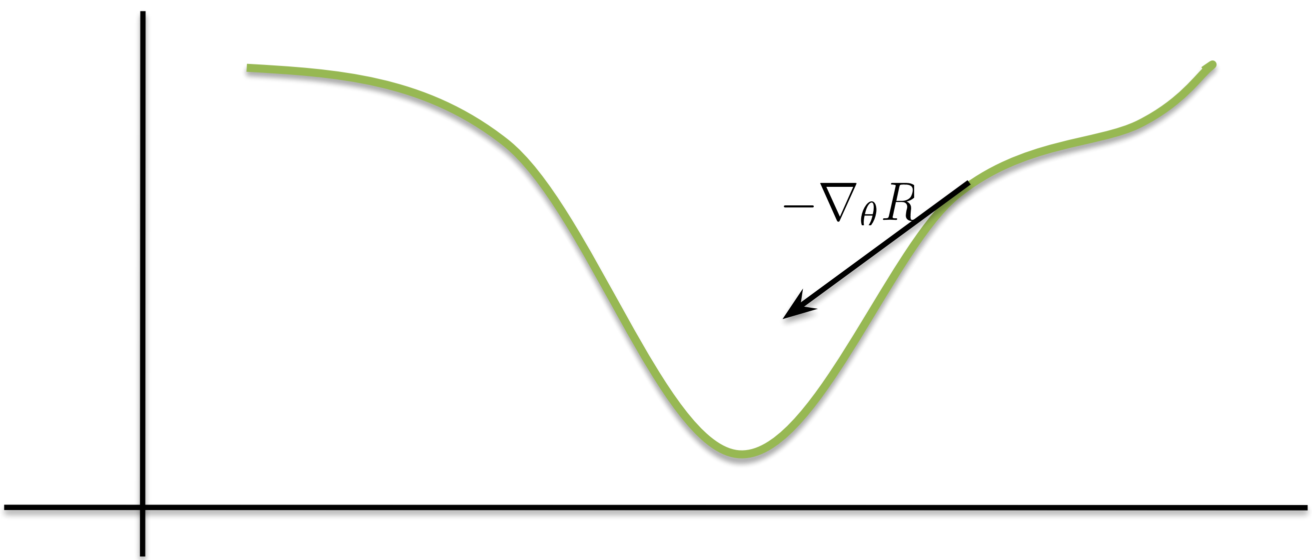
Gradient Descent

- Gradient points in the direction of fastest increase
- To minimize R, move in the opposite direction



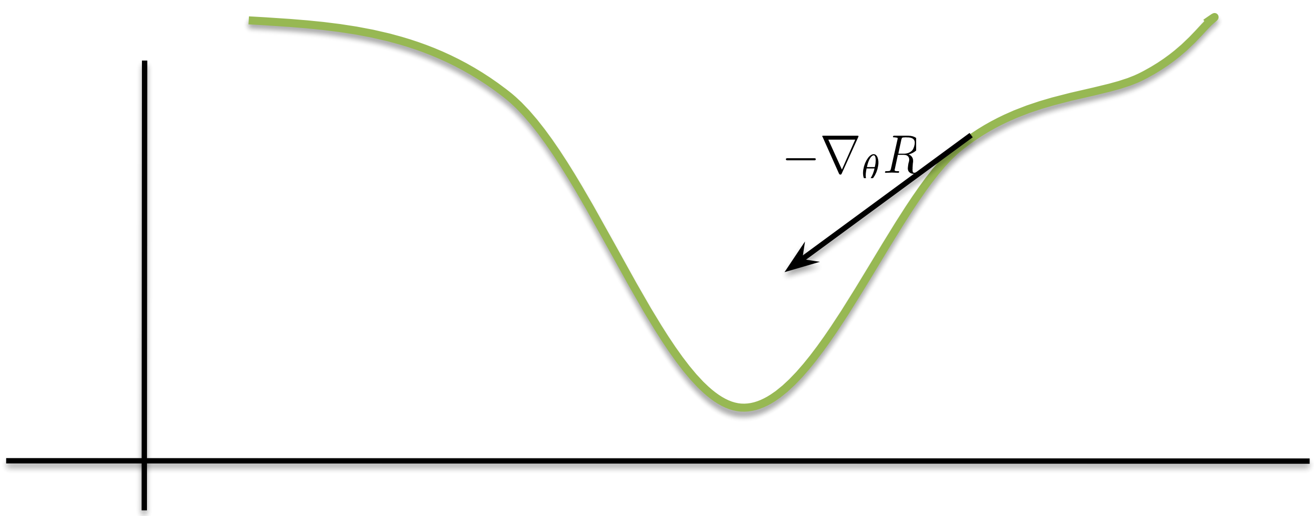
Gradient Descent

- Gradient points in the direction of fastest increase
- To minimize R, move in the opposite direction



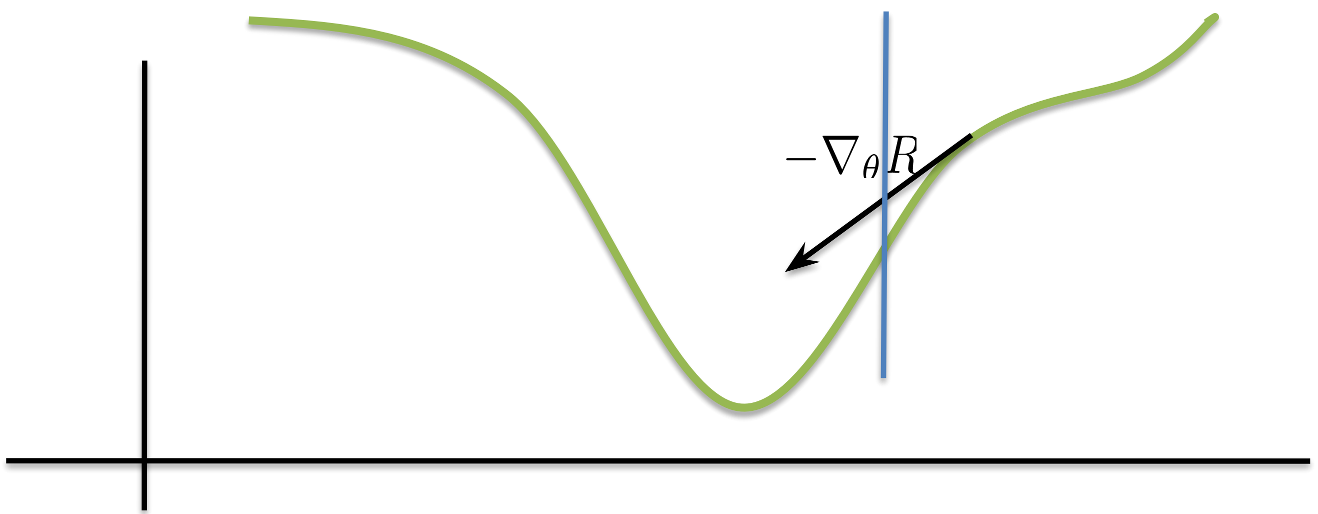
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- (nearly) guaranteed to converge to the minimum



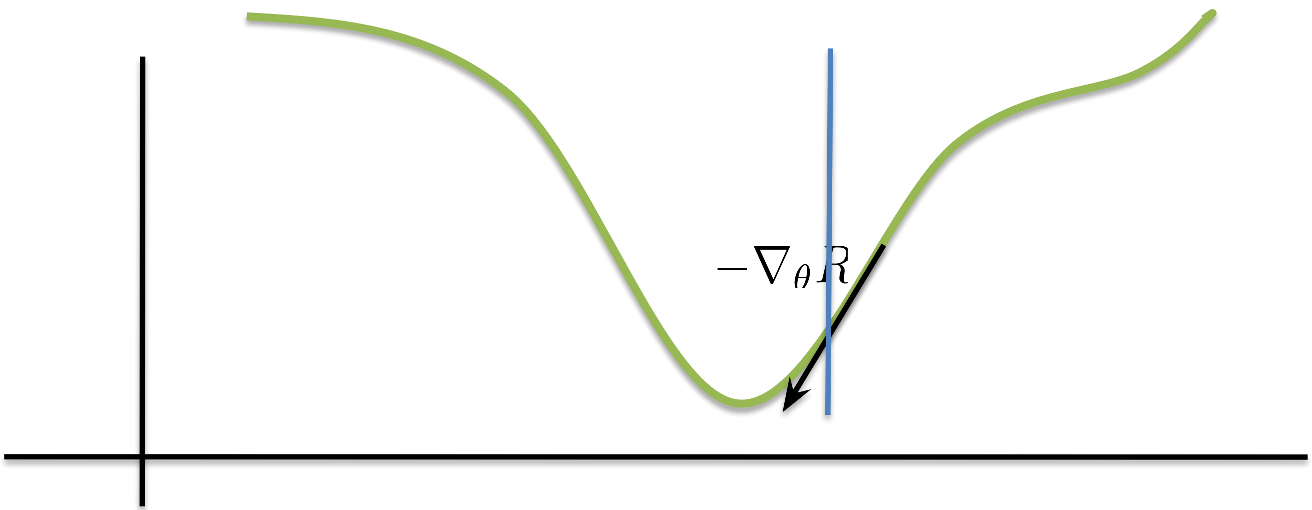
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- (nearly) guaranteed to converge to the minimum



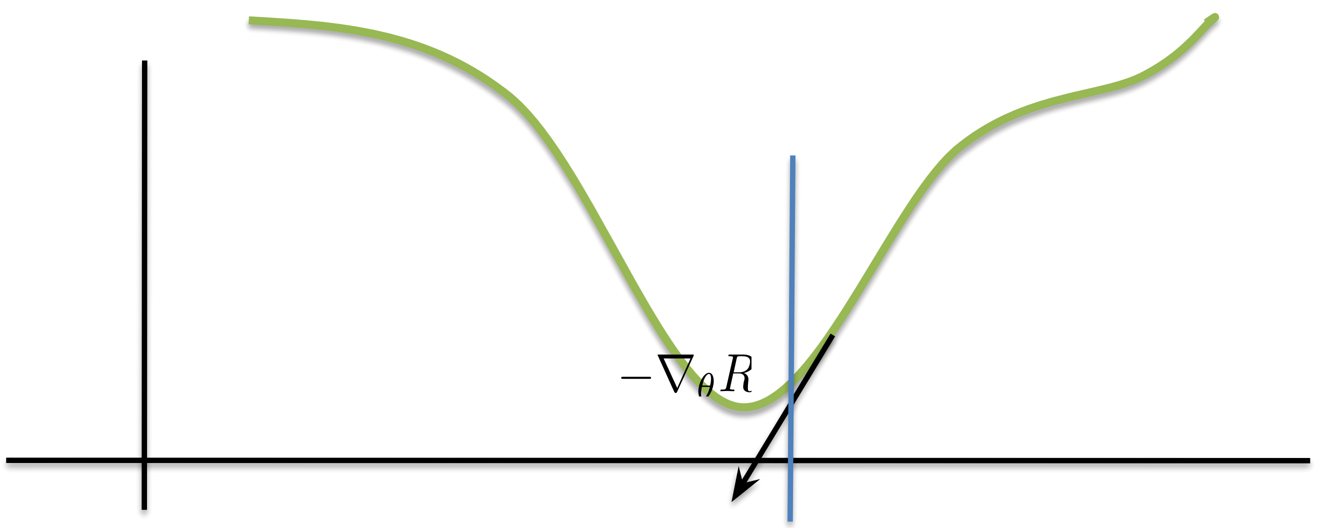
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- (nearly) guaranteed to converge to the minimum



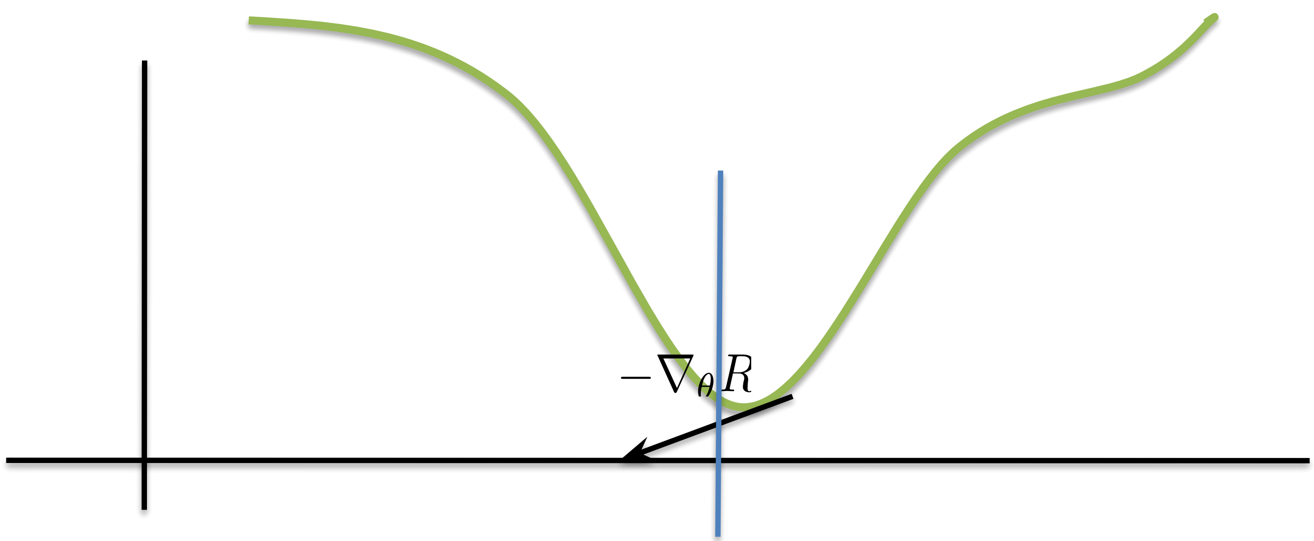
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- (nearly) guaranteed to converge to the minimum



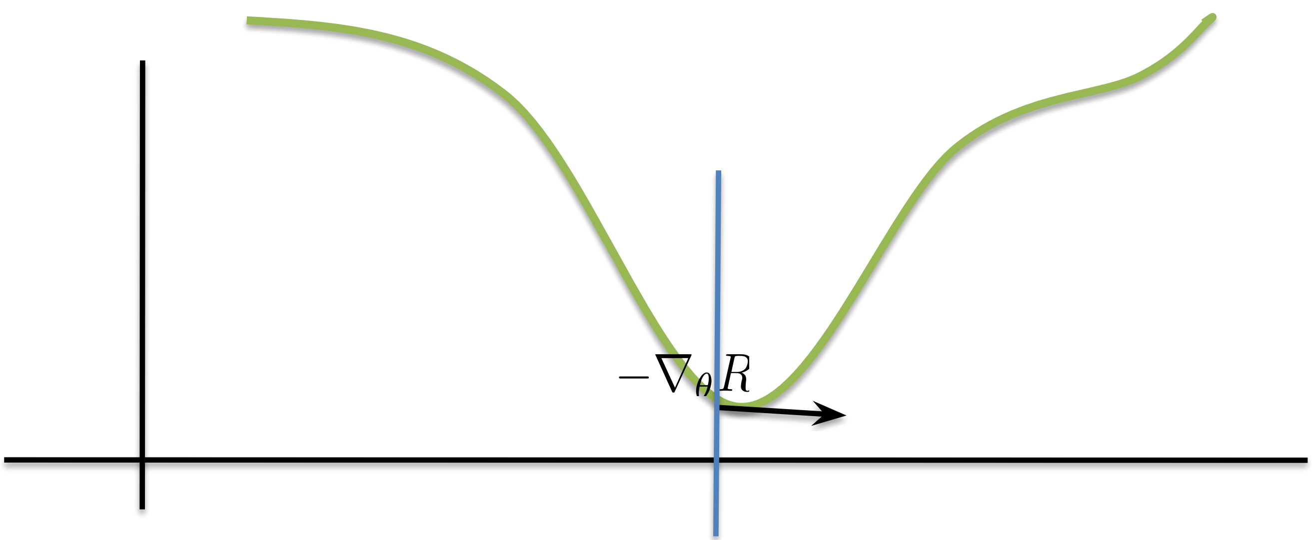
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- (nearly) guaranteed to converge to the minimum



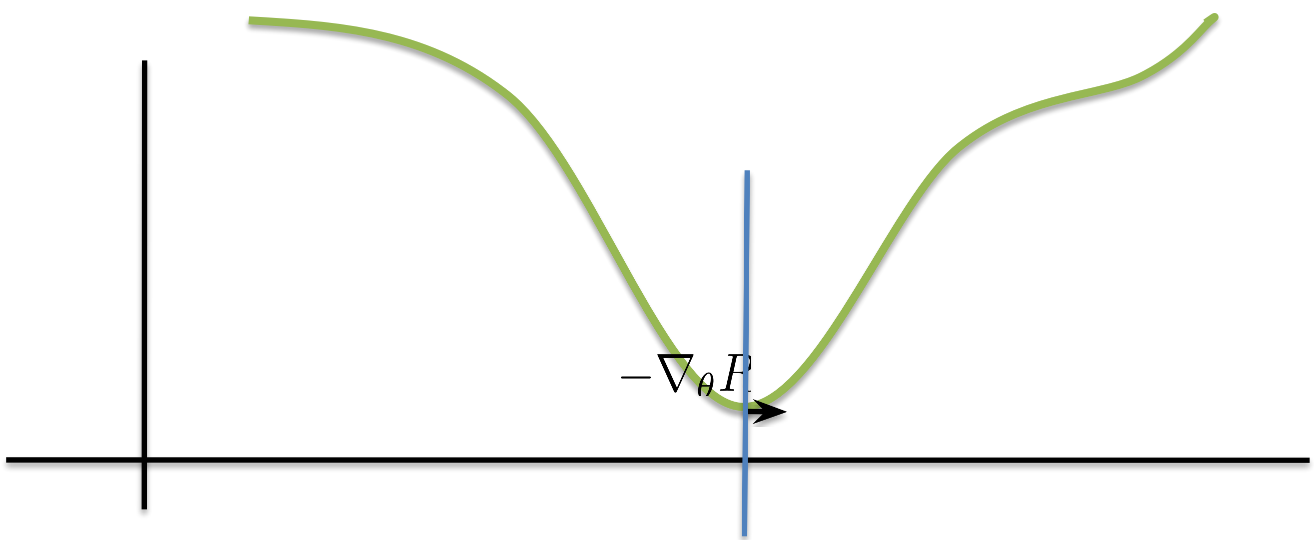
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- (nearly) guaranteed to converge to the minimum



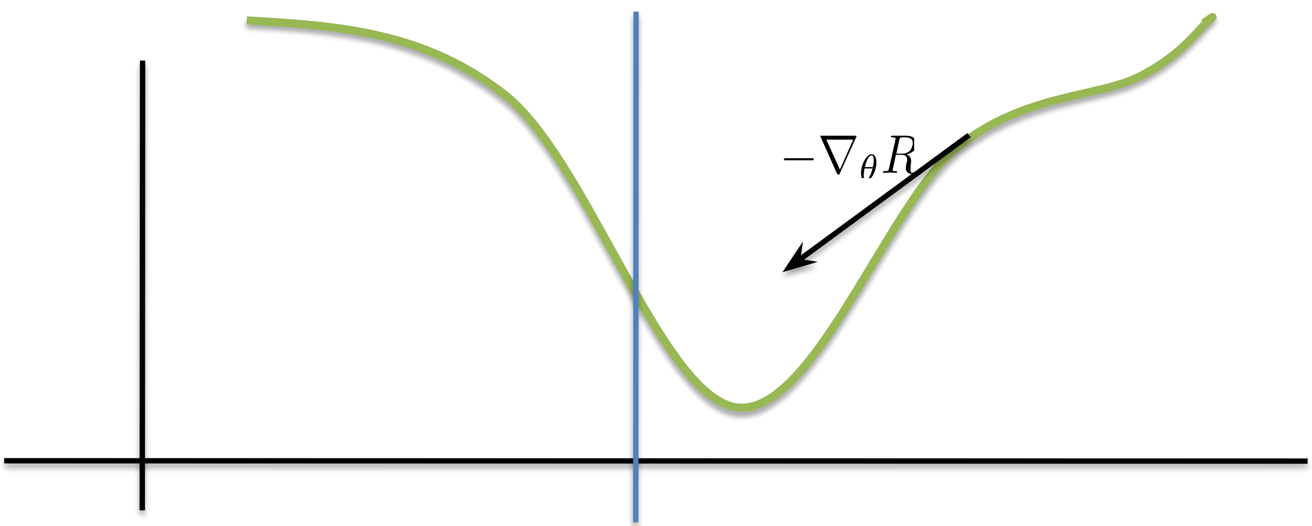
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- (nearly) guaranteed to converge to the minimum



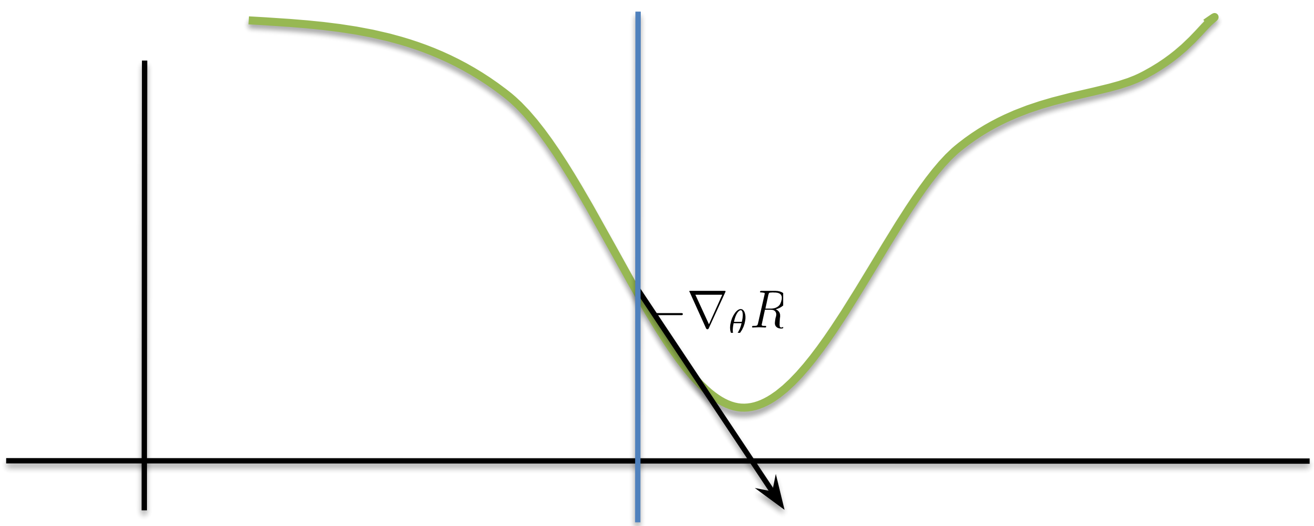
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



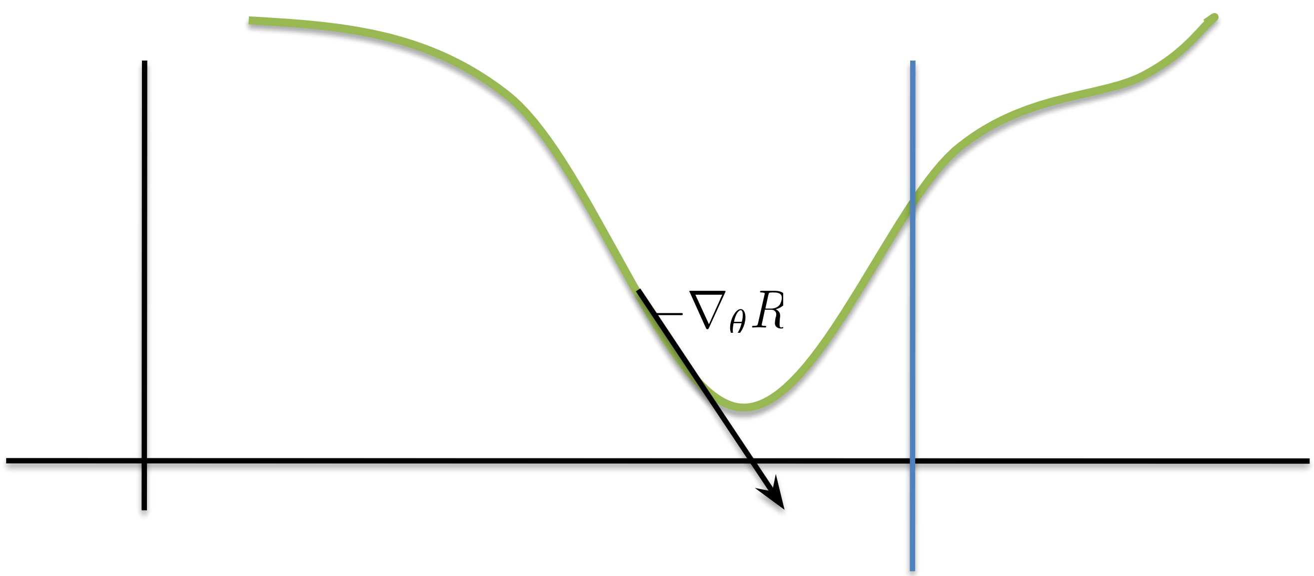
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



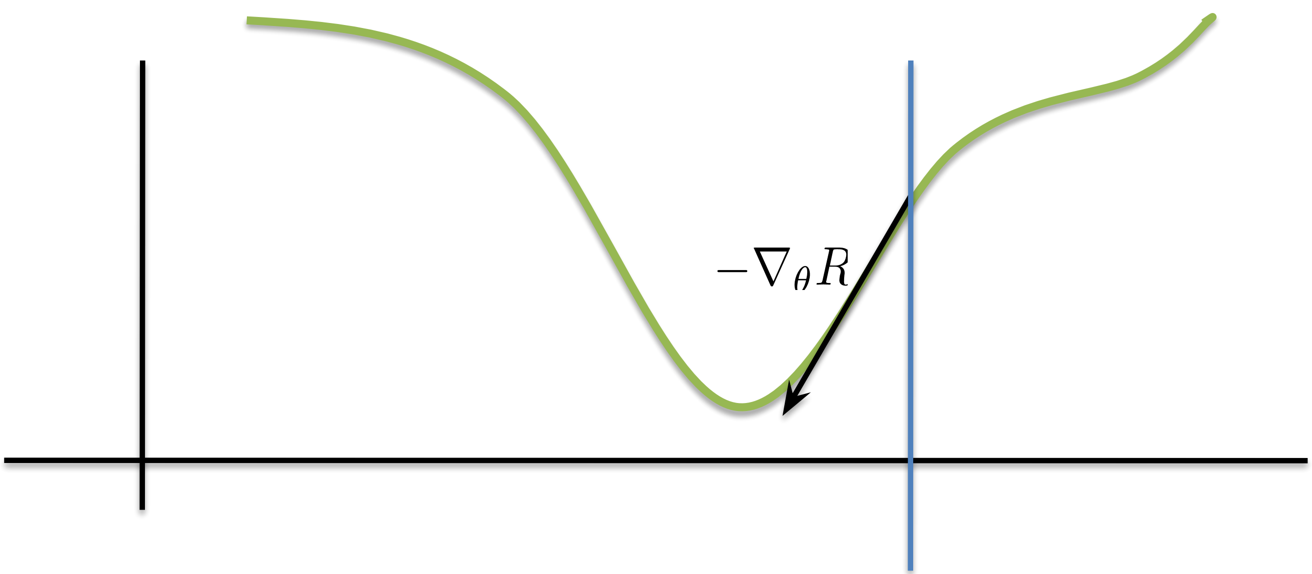
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



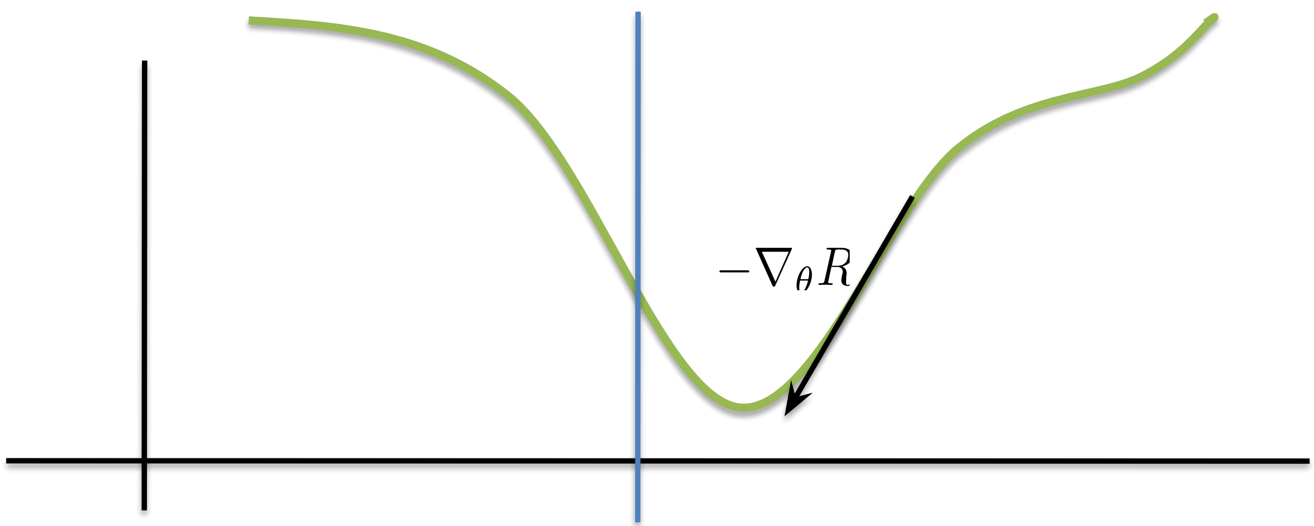
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



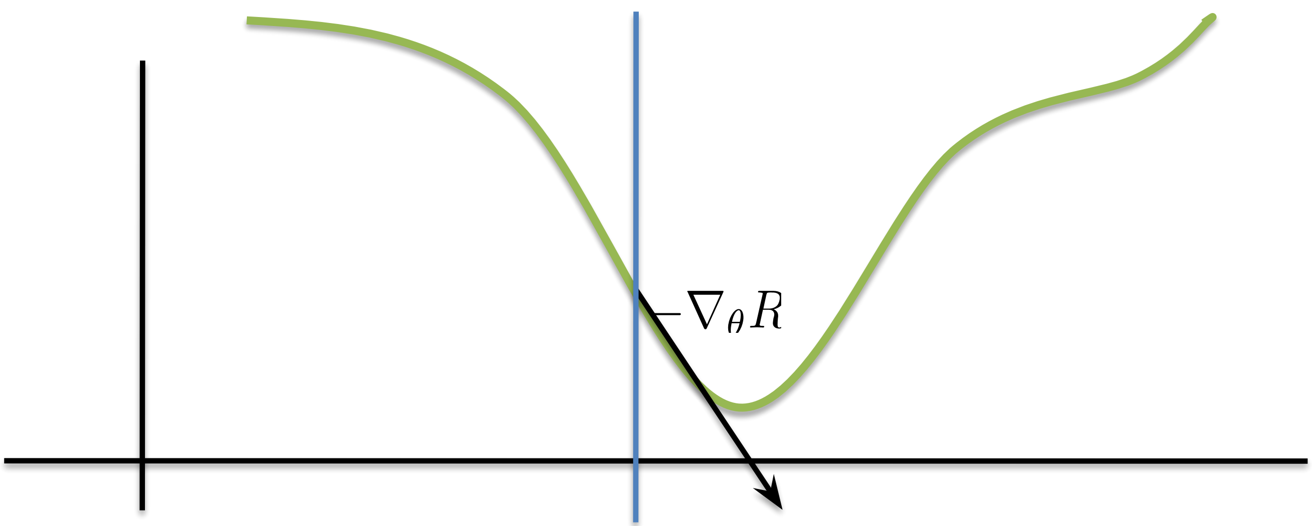
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



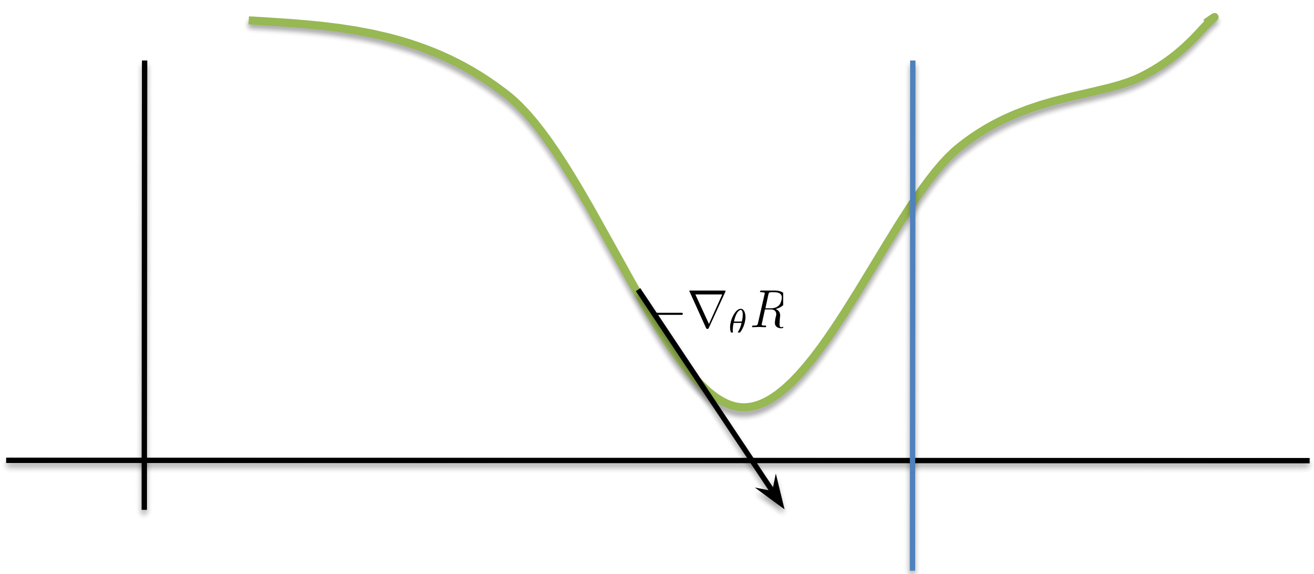
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



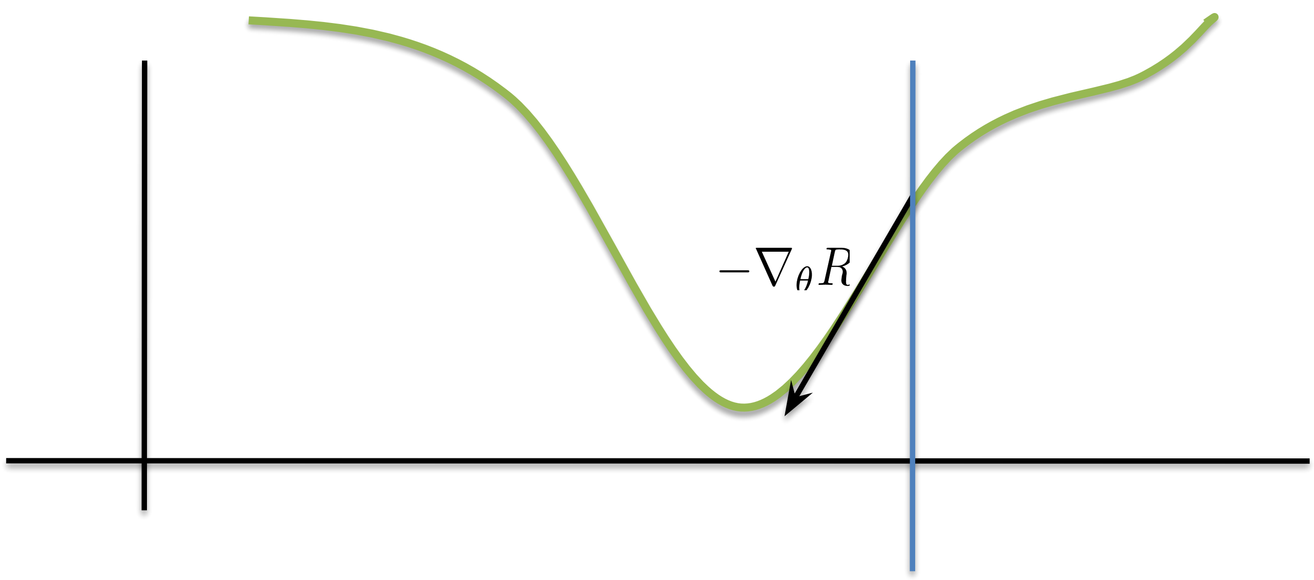
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



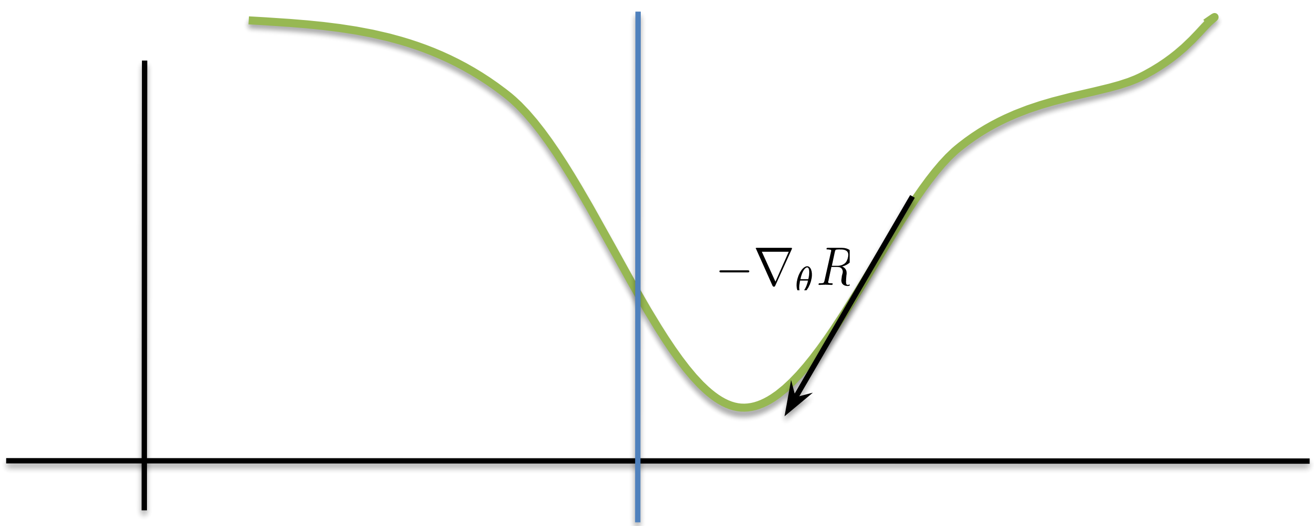
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



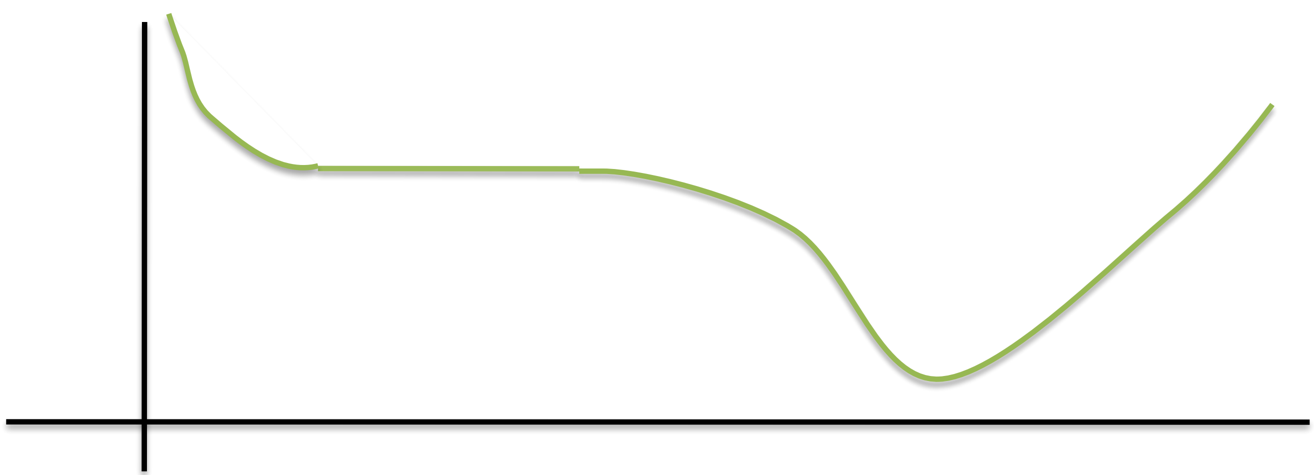
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can oscillate if η is too large



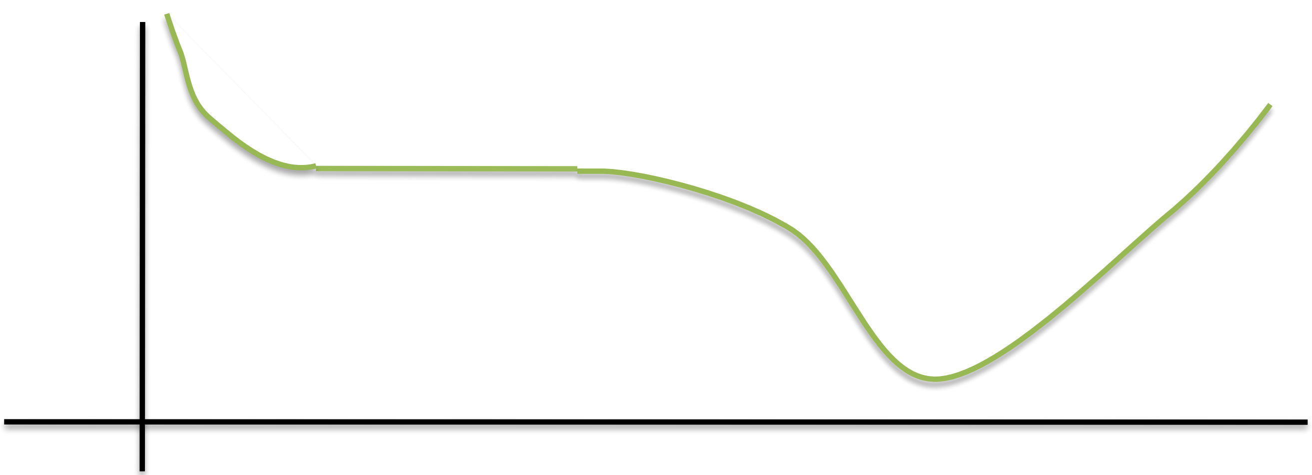
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can stall if $-\nabla_\theta R$ is ever 0 not at the minimum



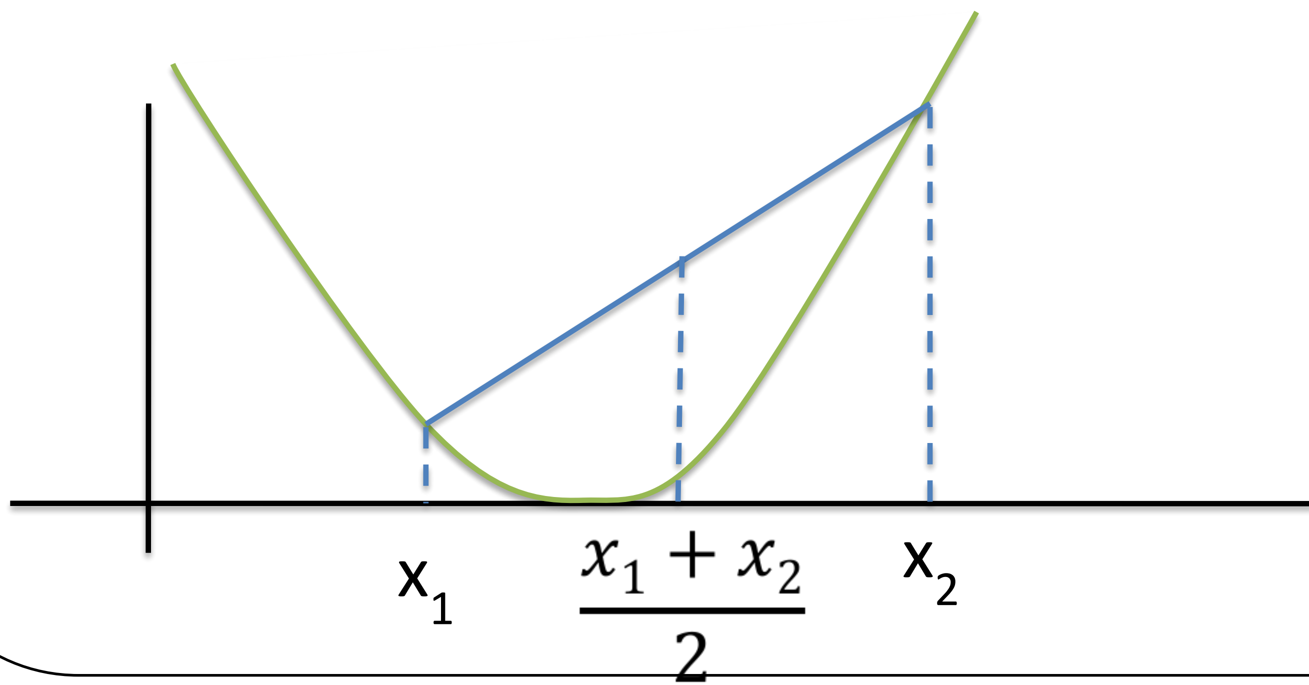
Gradient Descent

- Initialize Randomly $\theta_0 = random$
- Update with small steps $\theta_{t+1} = \theta_t - \eta \nabla_\theta R|_{\theta_t}$
- Can stall if $-\nabla_\theta R$ is ever 0 not at the minimum



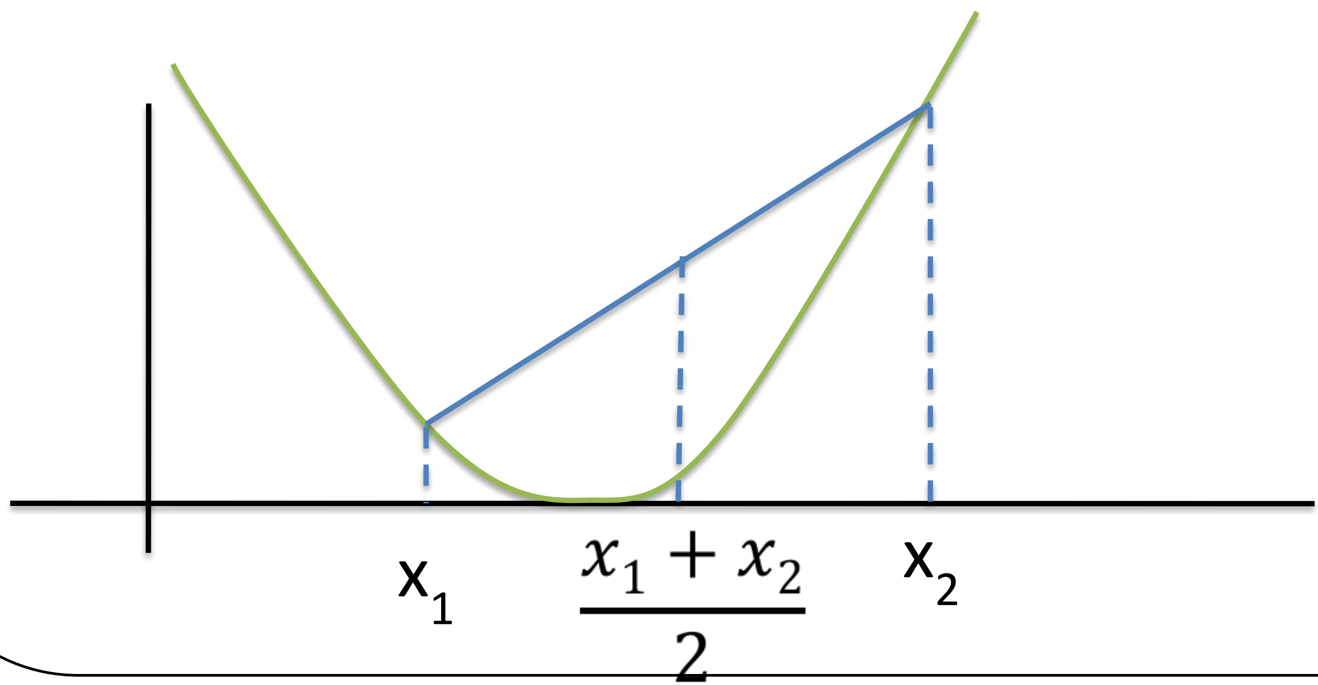
Convex function

- f(x) is convex if and only if $\forall\ x_1, x_2$
$$f\left(\frac{x_1 + x_2}{2}\right) \leq \frac{f(x_1) + f(x_2)}{2}$$
- Convex iff second derivative always positive
- Concave (the opposite definition)



Convex Optimization

- If stop gradient descent, has to be the optimum
- In Newton method, use second derivative to decide step size



Multi-label Logistic Regression

- K labels $p(\mathcal{C}_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$ $a_k = \mathbf{w}_k^{\mathrm{T}}\phi.$

- Key: $t_{nk} = t_n{}^k \in \{0, 1\}$, Let $y_{nk} = y^k(x_n)$

$$p(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(\mathcal{C}_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}}$$

$$E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\, \phi_n$$

Classification approaches

$$p(c_j|\vec{x}) = \frac{p(\vec{x}|c_j)p(c_j)}{p(\vec{x})}$$

- **Generative**
  - Models the joint distribution between c and x (p(x,c))
  - E.g. LDA, QDA, even Naive Bayes, GMM, HMM
  - High number of parameters to estimate
  - High data requirements
- **Discriminative**
  - Fewer parameters to estimate
  - Easier, often perform better
  - Learn by optimizing conditional likelihood, or simply an error function
  - Posterior probability or simply a discriminant function
  - E.g. Logistic regression, Perceptron, Decision tree, random forest, KNN, etc.
  - Issue: only learned a classifier, nothing else

$$p(c_j|\vec{x})$$

$$f(\vec{x}) = c_j$$