Deep Learning for Graphics

Machine Learning Introduction

Niloy Mitra	Iasonas Kokkinos	Paul Guerrero	Vladimir Kim	Kostas Rematas	Tobias Ritschel
UCL	UCL/Facebook	UCL	Adobe Research	U Washington	UCL

Course Overview

- **Part I: Introduction and ML Basics**
- Part II: Supervised Neural Networks: Theory and Applications
- Part III: Unsupervised Neural Networks: Theory and Applications
- Part IV: Beyond Image Data

Machine Learning

Machine Learning

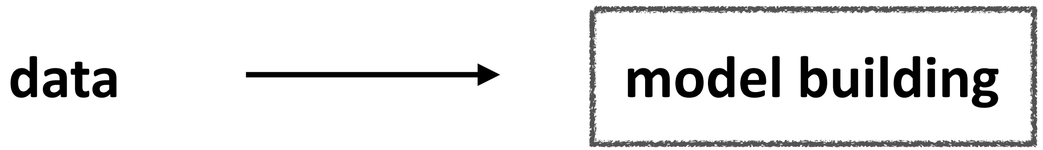
Machine learning is a field of **computer science** that uses statistical techniques to give computer systems the ability to ***learn*** (i.e., progressively improve performance on a specific task) with **data**, without being explicitly programmed.

'ML' coined by Arthur Samuel, 1959.

Machine Learning

Machine learning is a field of **computer science** that uses statistical techniques to give computer systems the ability to ***learn*** (i.e., progressively improve performance on a specific task) with **data**, without being explicitly programmed.

'ML' coined by Arthur Samuel, 1959.



Machine Learning

Machine learning is a field of **computer science** that uses statistical techniques to give computer systems the ability to ***learn*** (i.e., progressively improve performance on a specific task) with **data**, without being explicitly programmed.

'ML' coined by Arthur Samuel, 1959.

Machine Learning Variants

- **Supervised**
  - Classification
  - Regression
  - Data consolidation
- **Unsupervised**
  - Clustering
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**

  Some data supervised, some unsupervised
- **Reinforcement learning**

  Supervision: sparse reward for a sequence of decisions

Machine Learning Variants

- **Supervised**
  - **Classification**
  - Regression
  - Data consolidation
- **Unsupervised**
  - Clustering
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**

  Some data supervised, some unsupervised
- **Reinforcement learning**

  Supervision: sparse reward for a sequence of decisions

Classification Examples

- Digit Recognition

Classification Examples

- Digit Recognition

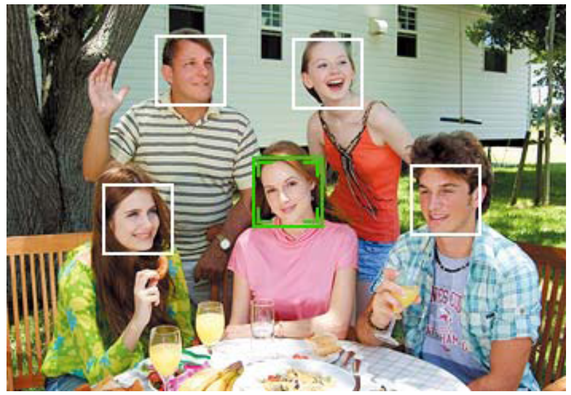
- Spam Detection

Classification Examples

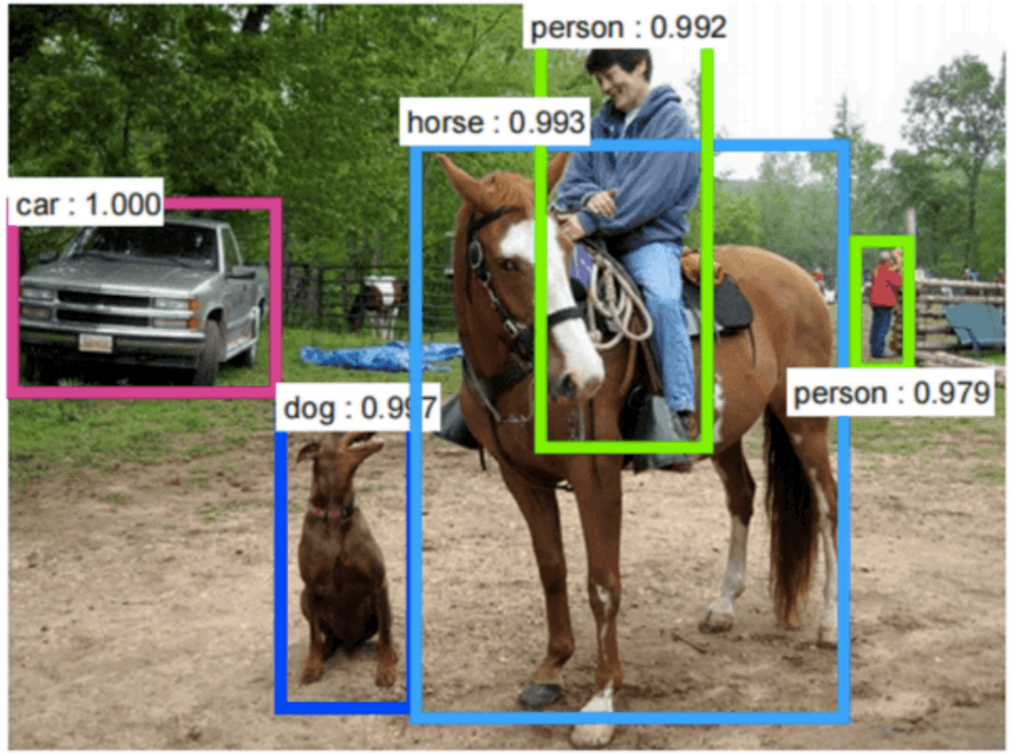
- Digit Recognition

- Spam Detection

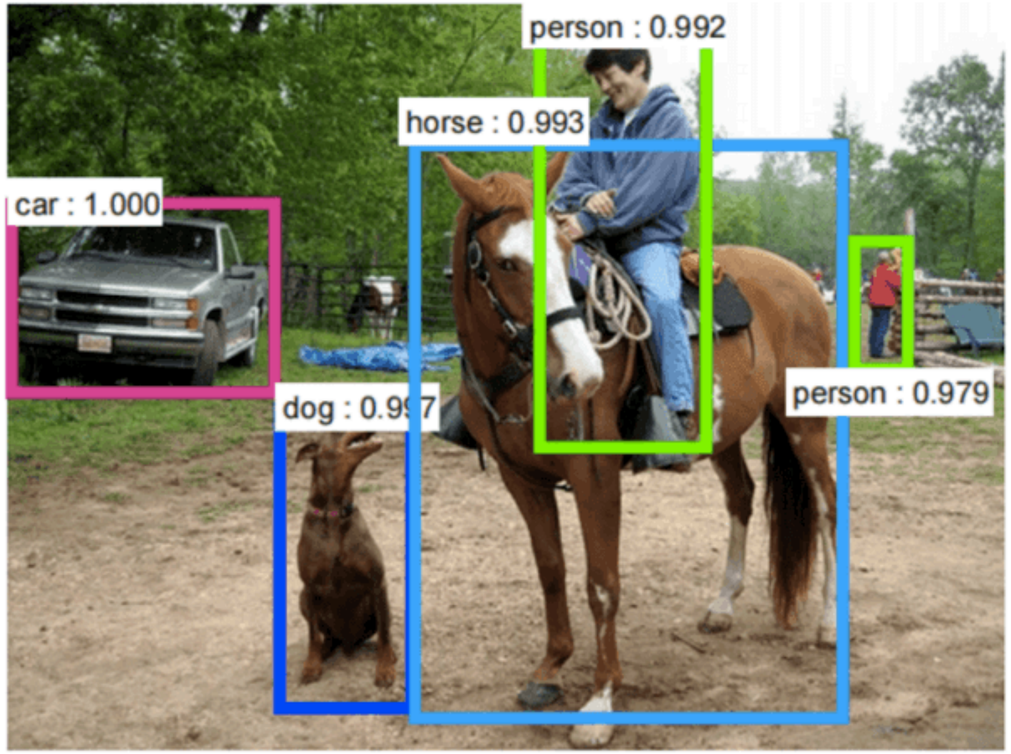
- Face detection

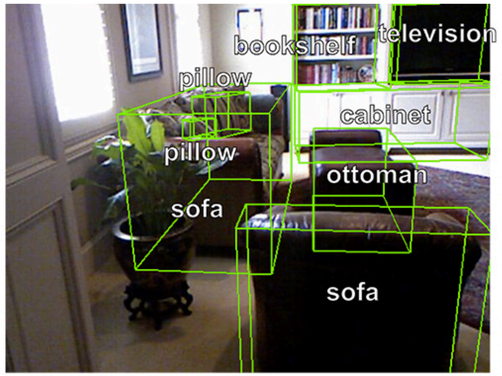


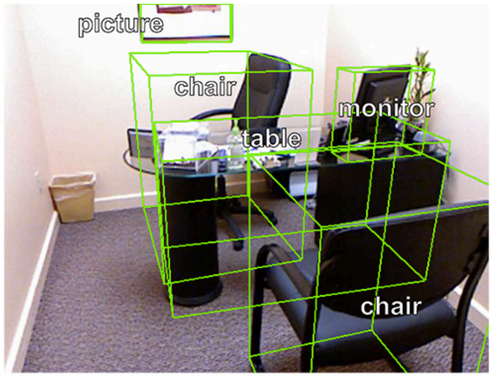
Segmentation + Classification in Real Images

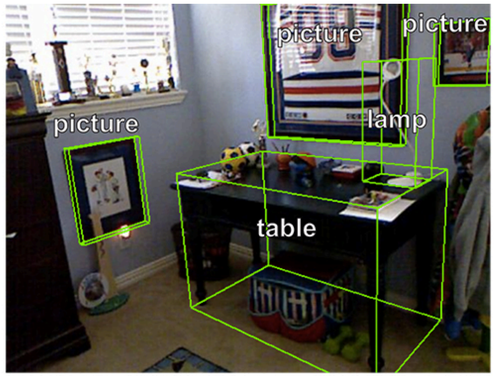


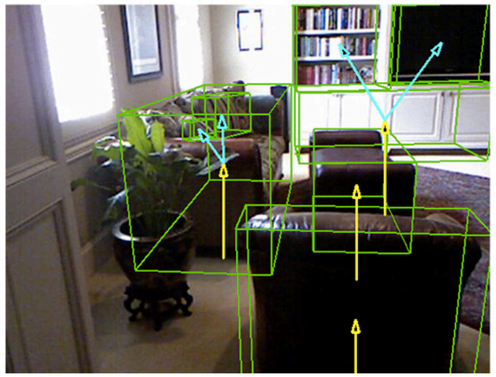
Segmentation + Classification in Real Images

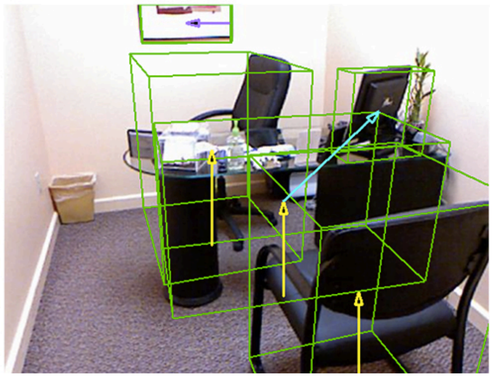


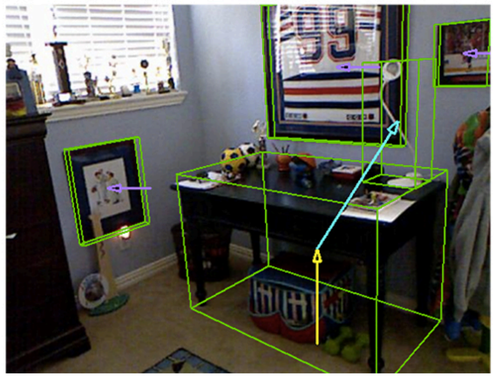




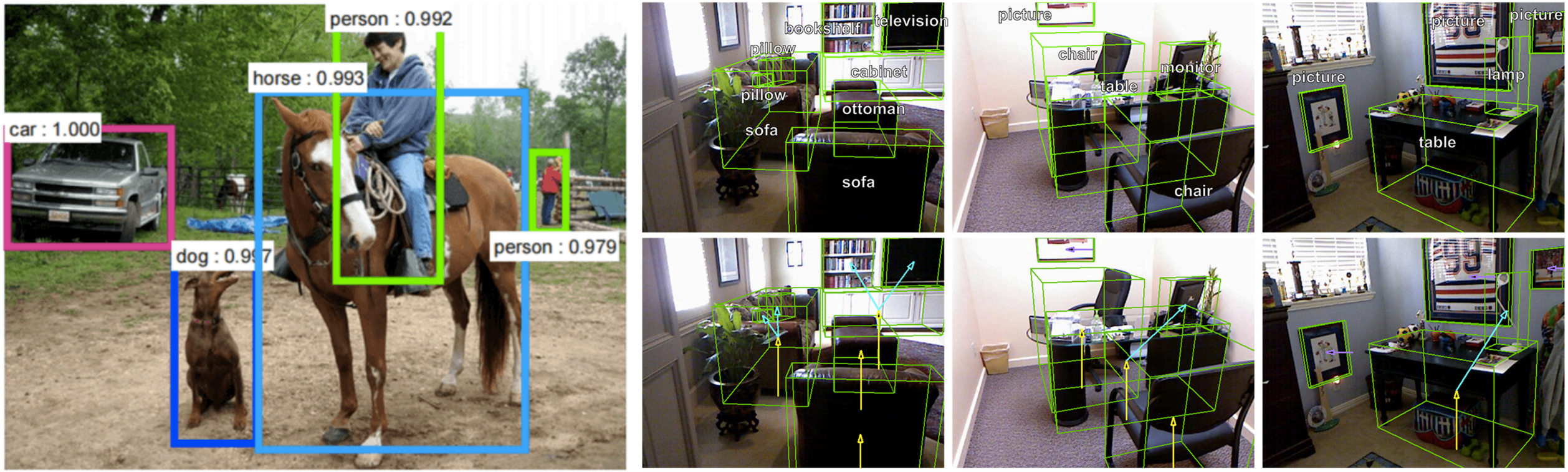








Segmentation + Classification in Real Images



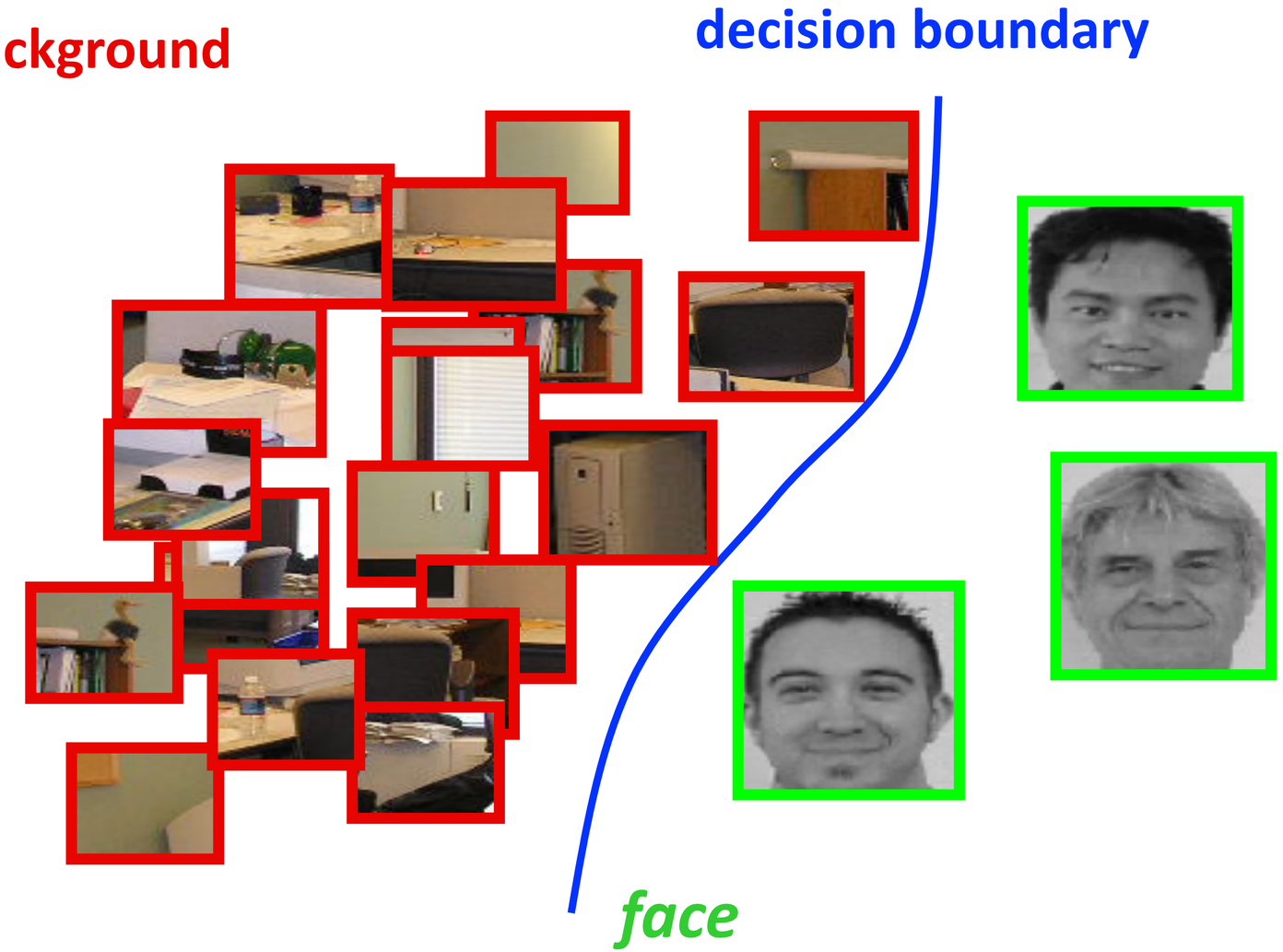
Evaluation measures: Confusion matrix, ROC curve, precision, recall, etc.

`Faceness' Function: Classifier

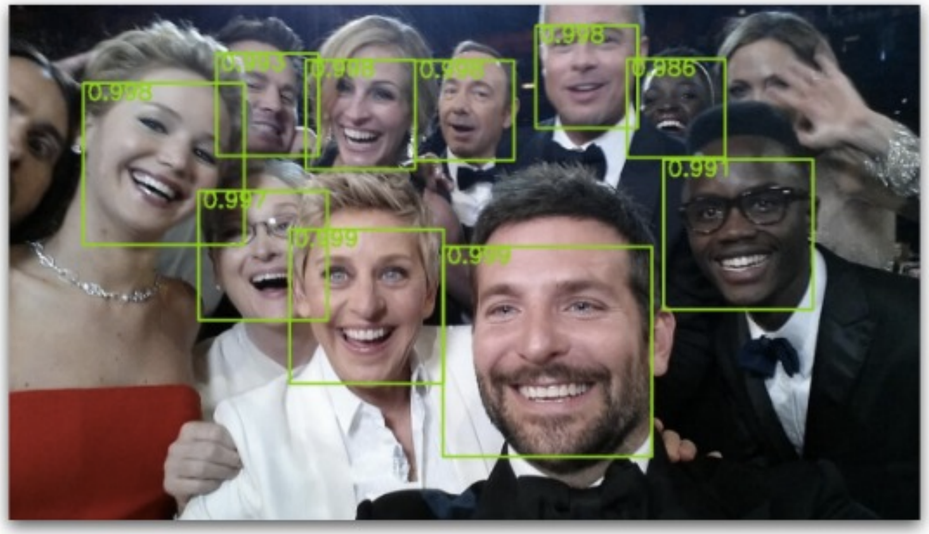
background

face

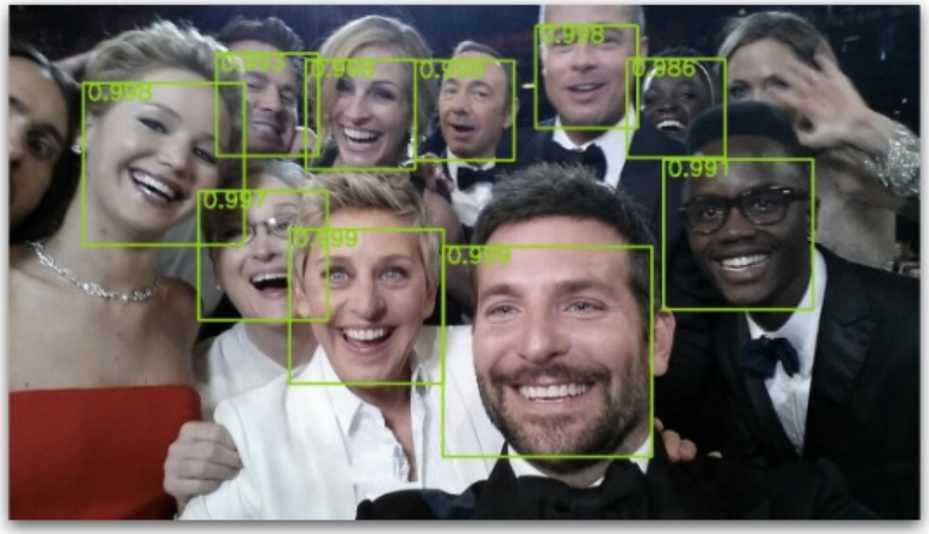
`Faceness' Function: Classifier



Face Detection



Face Detection



CMU Science Lab

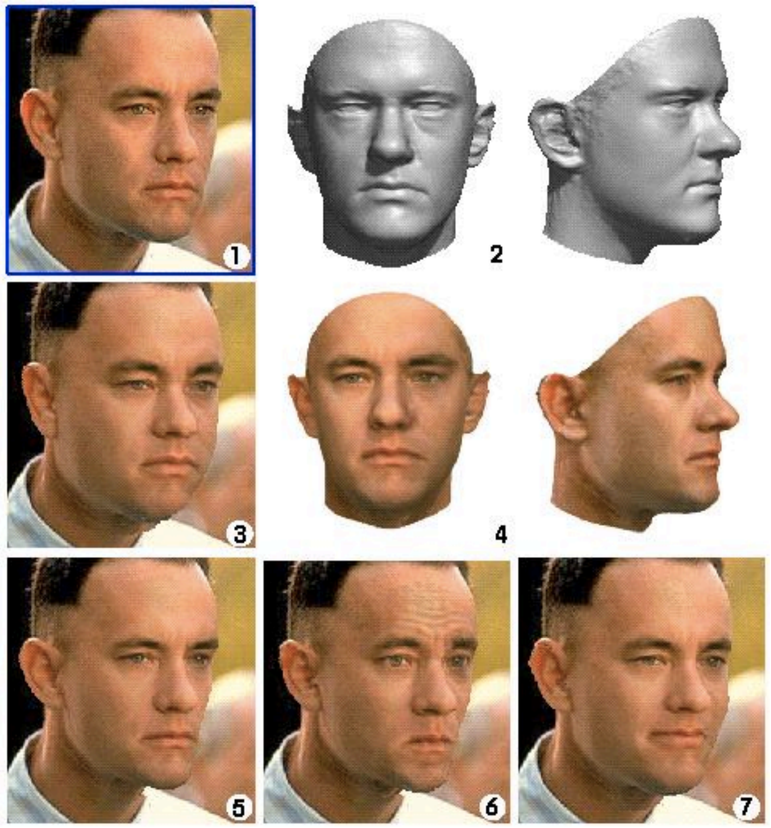
Machine Learning Variants

- **Supervised**
  - Classification
  - **Regression**
  - Data consolidation
- **Unsupervised**
  - Clustering
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**

  Some data supervised, some unsupervised
- **Reinforcement learning**

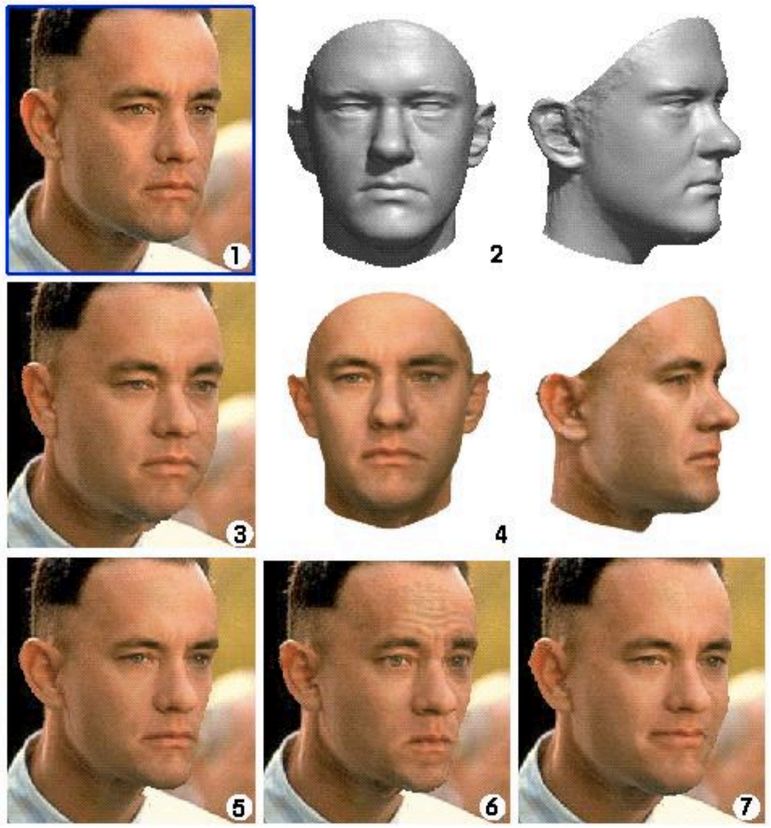
  Supervision: sparse reward for a sequence of decisions

Human Face/Pose Estimation

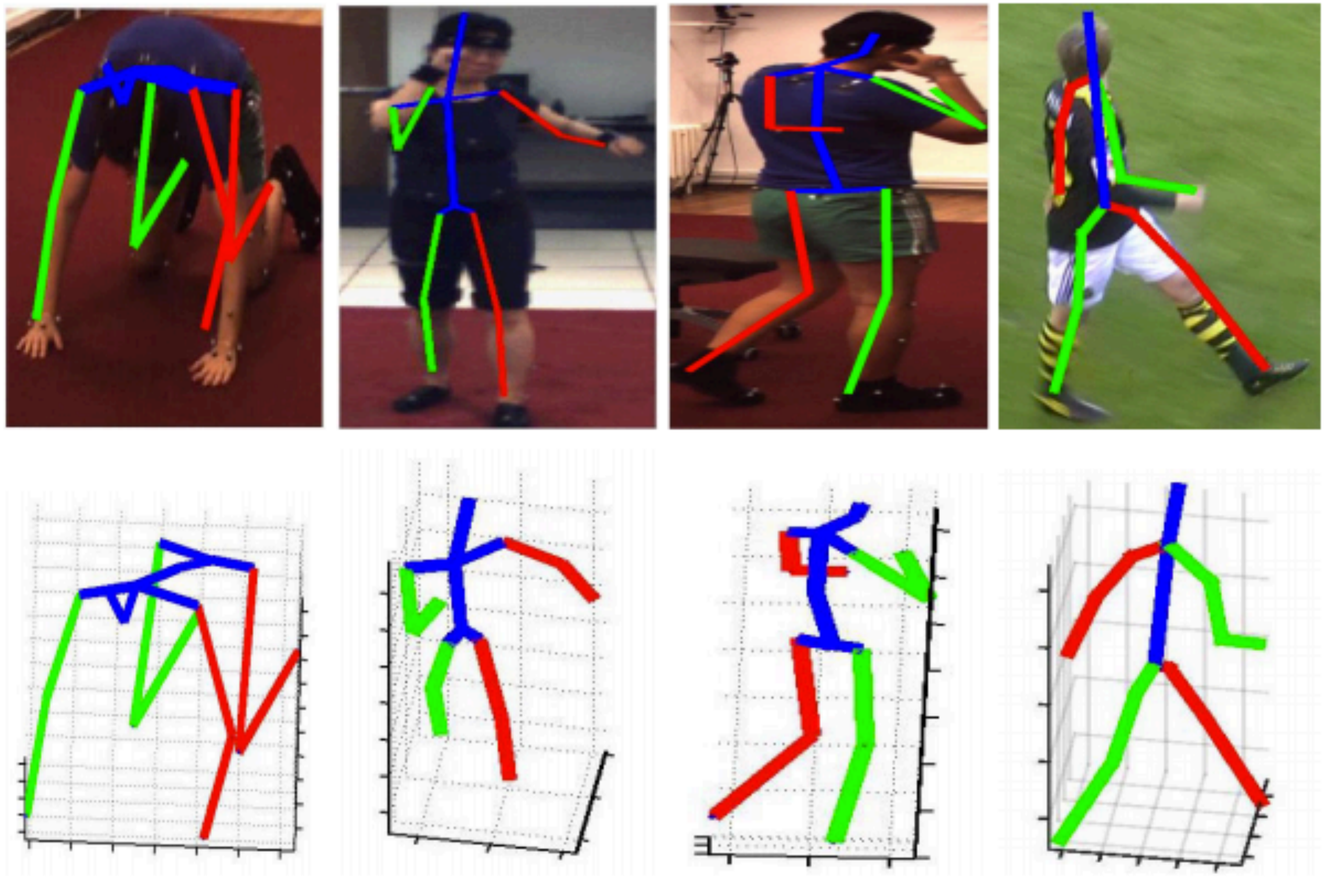


[Blanz and Vetter, Siggraph, 1999]

Human Face/Pose Estimation

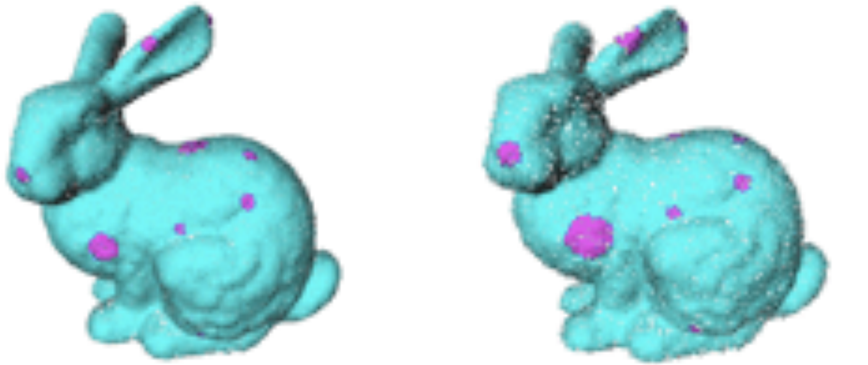


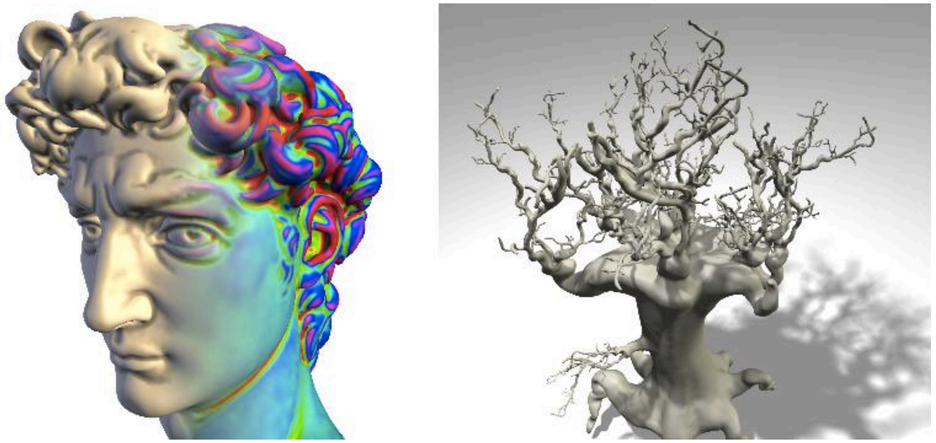
[Blanz and Vetter, Siggraph, 1999]



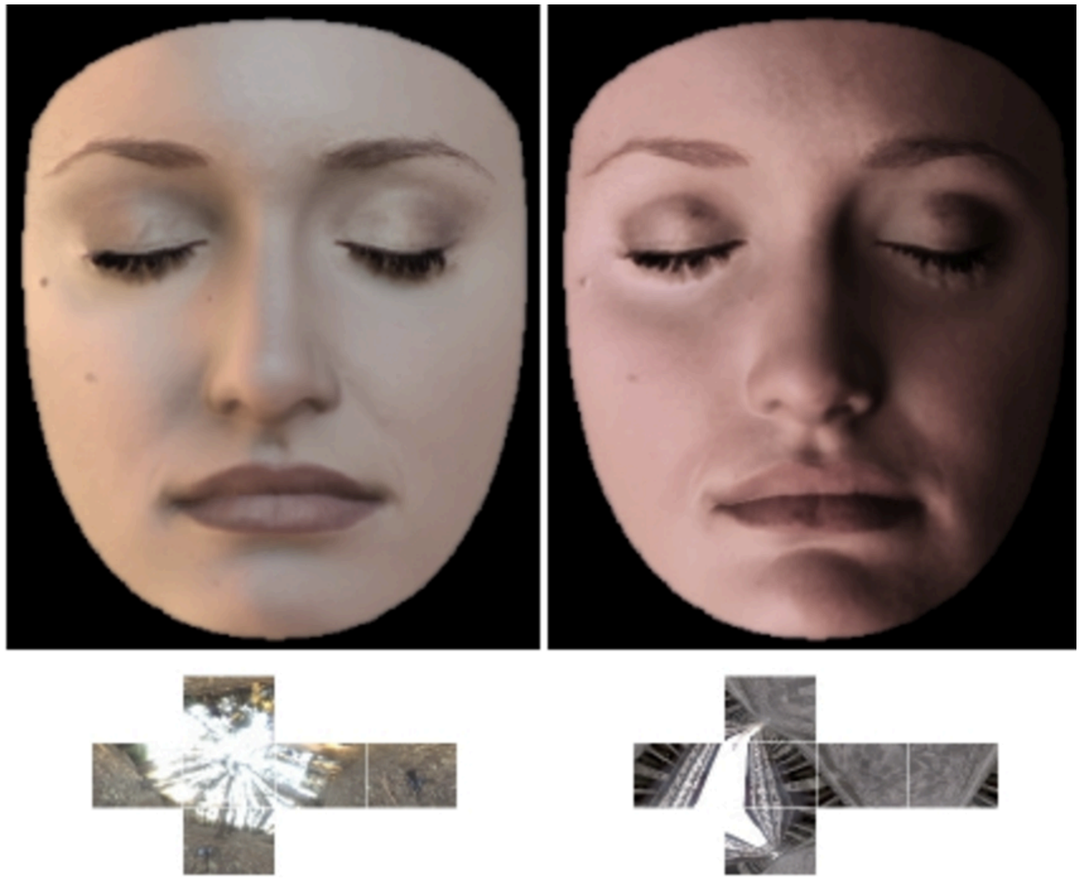
Regression: Model Estimation

[Mitra et al. SoCG, 2003]





[Guennebaud et al., Siggraph, 2007]



[Zwicker et al., EGSR, 2005]

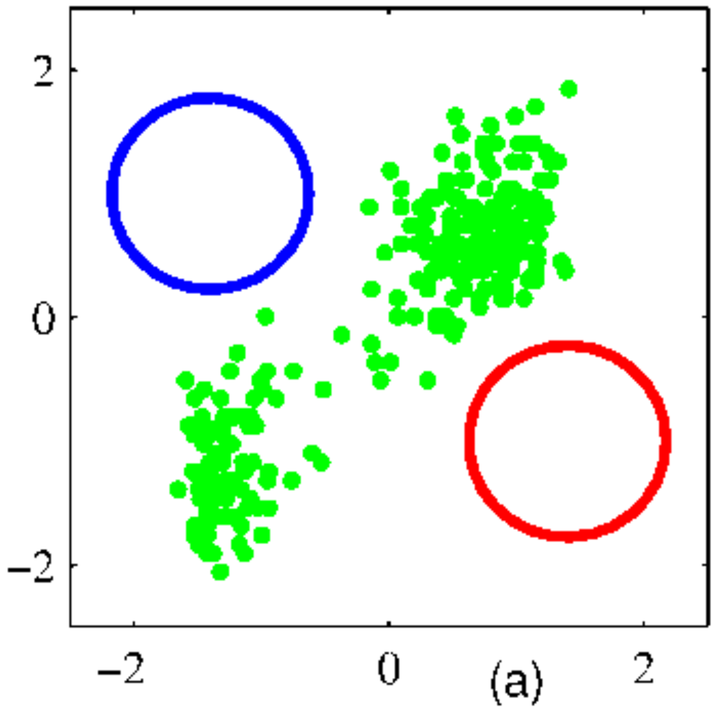
Machine Learning Variants

- **Supervised**
  - Classification
  - Regression
  - Data consolidation
- **Unsupervised**
  - **Clustering**
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**

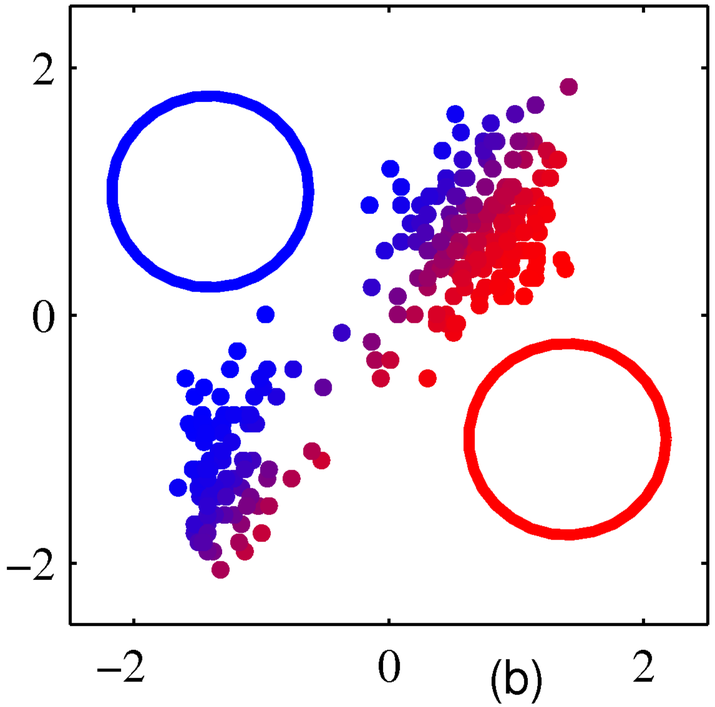
  Some data supervised, some unsupervised
- **Reinforcement learning**

  Supervision: sparse reward for a sequence of decisions

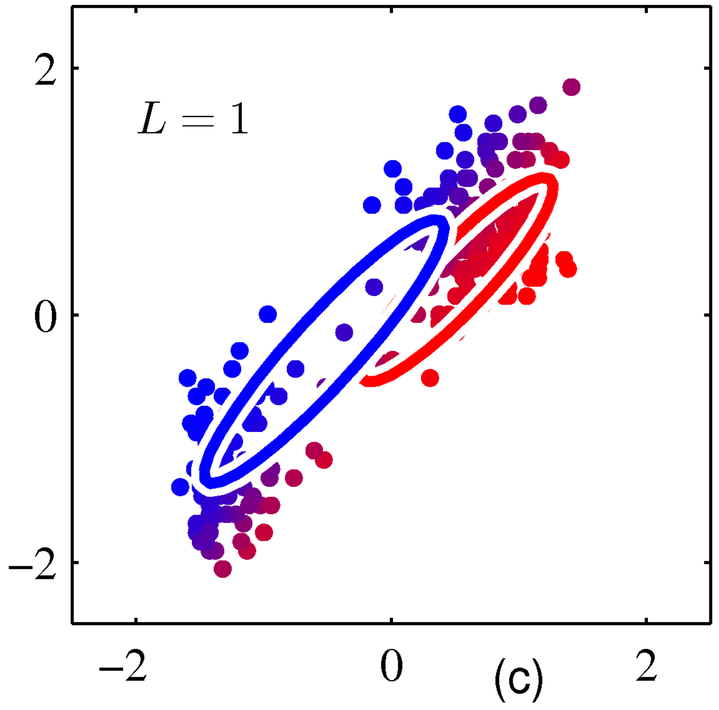
Clustering: Color Points According to X



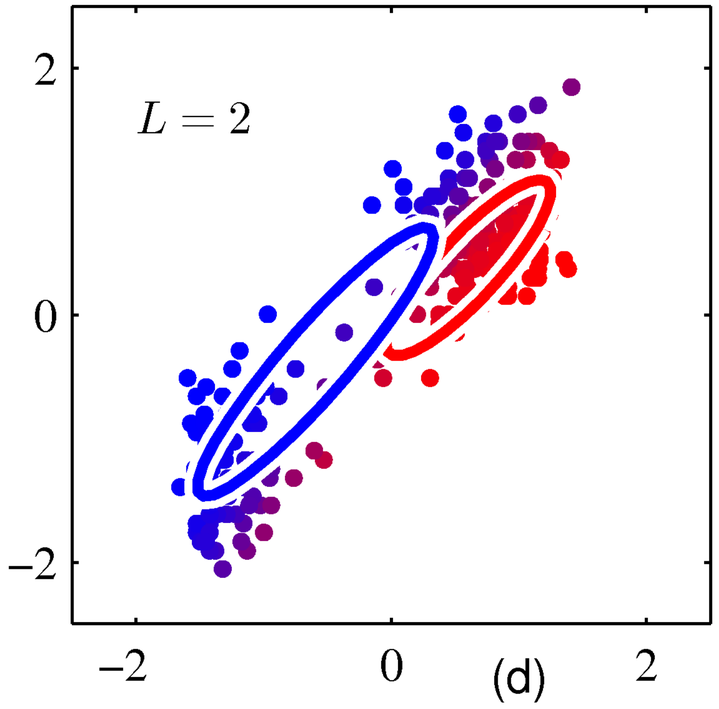
Clustering: Color Points According to X



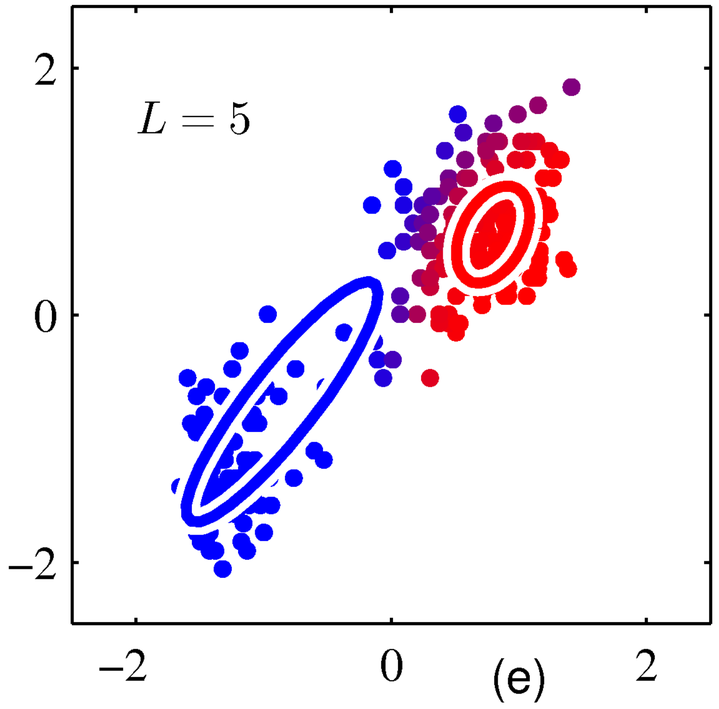
Clustering: Color Points According to X



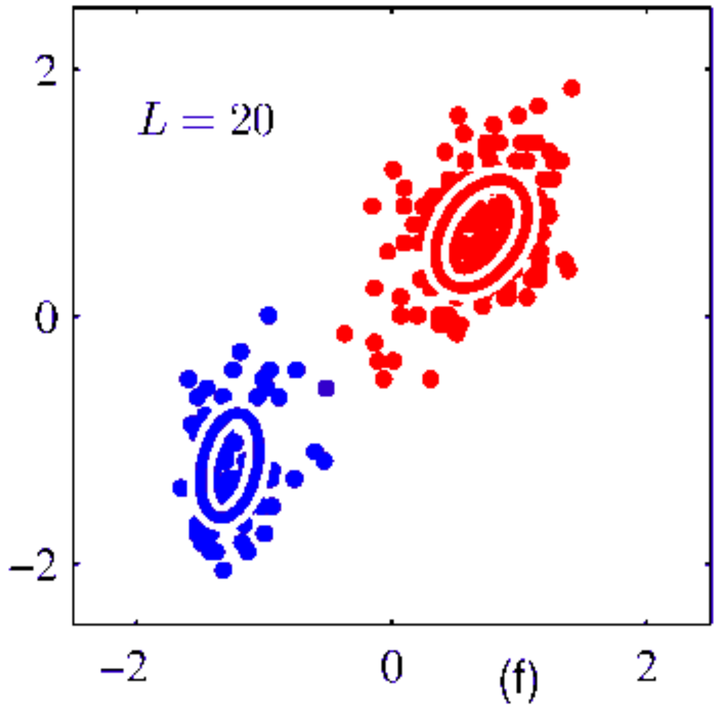
Clustering: Color Points According to X



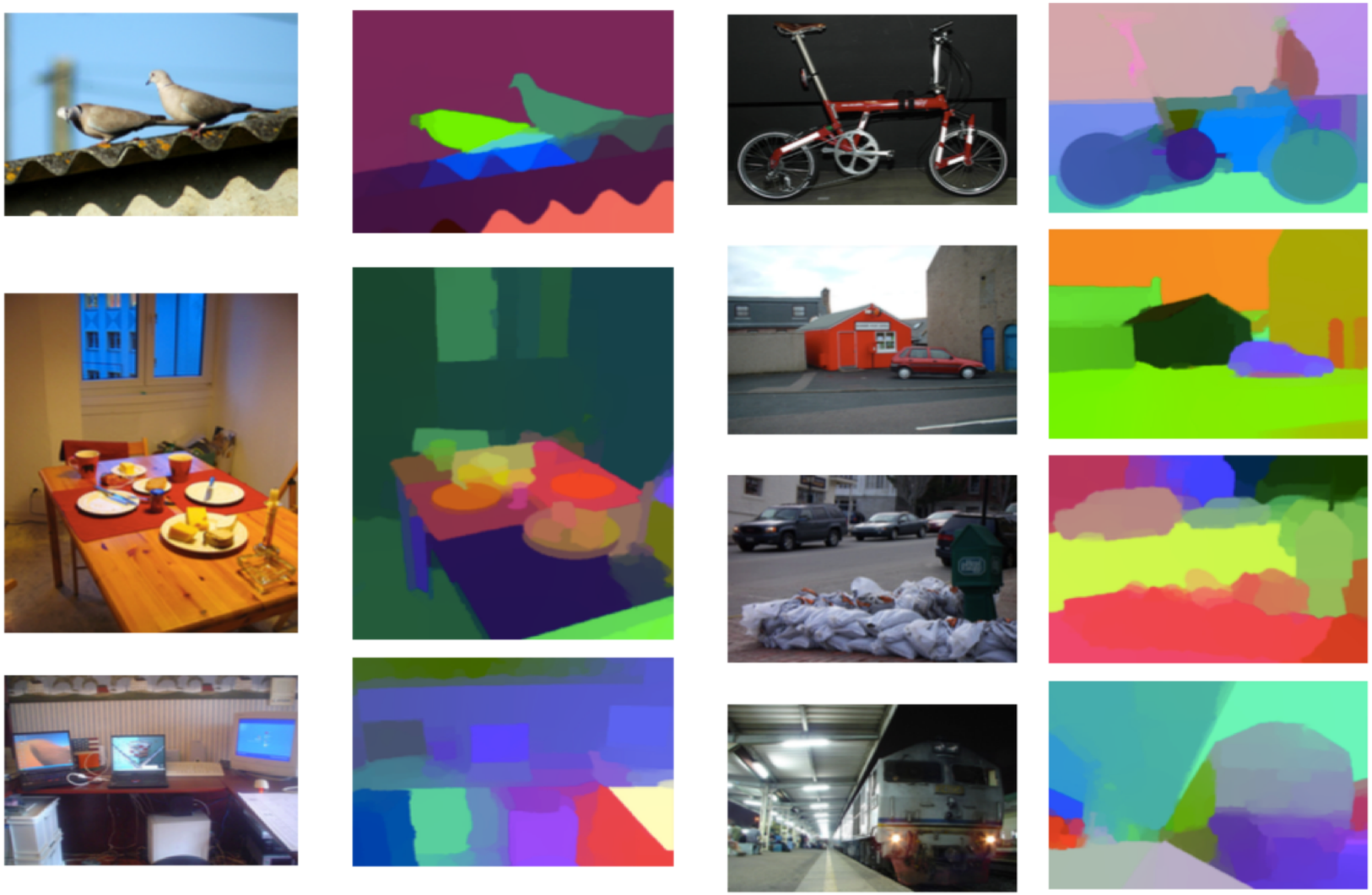
Clustering: Color Points According to X



Clustering: Color Points According to X

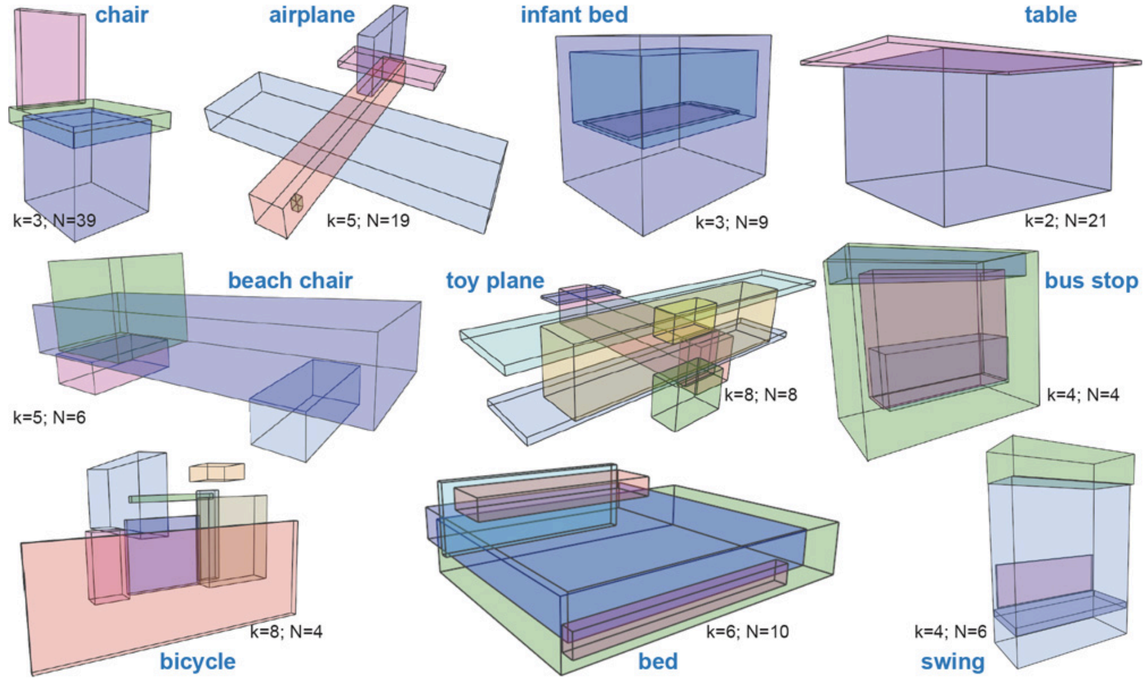


Clustering Examples: Image Segmentation using NCuts



Clustering Examples

[Chu et al., TVCG, 2009]



[Zheng et al., Eurographics, 2014]

Machine Learning Variants

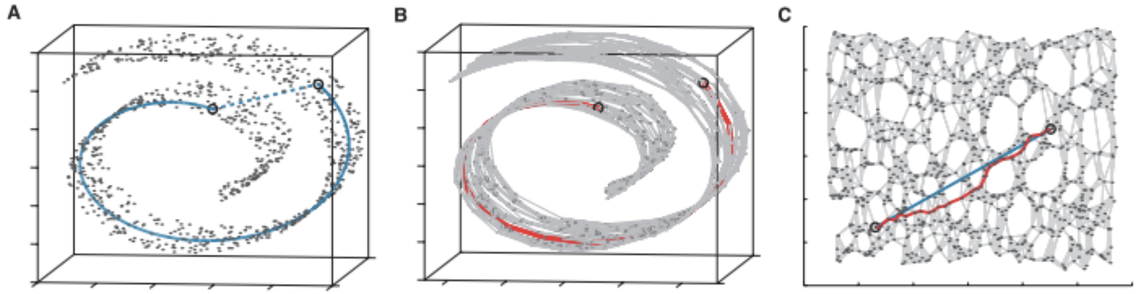
- **Supervised**
  - Classification
  - Regression
  - Data consolidation
- **Unsupervised**
  - Clustering
  - **Dimensionality Reduction**
- **Weakly supervised/semi-supervised**

  Some data supervised, some unsupervised
- **Reinforcement learning**

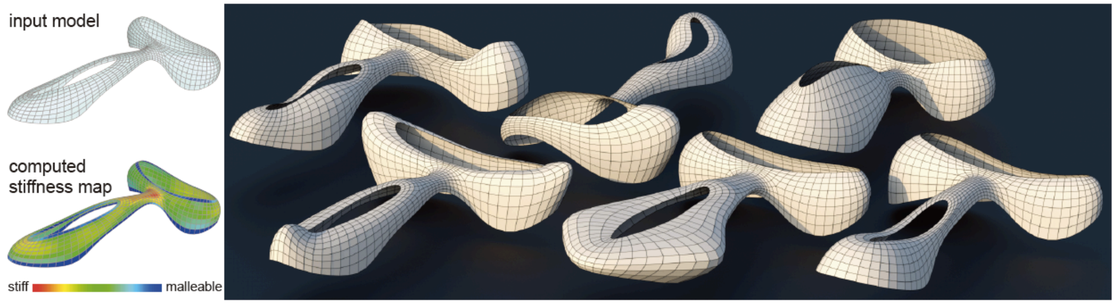
  Supervision: sparse reward for a sequence of decisions

Dimensionality Reduction (Manifold Learning)

Isomap

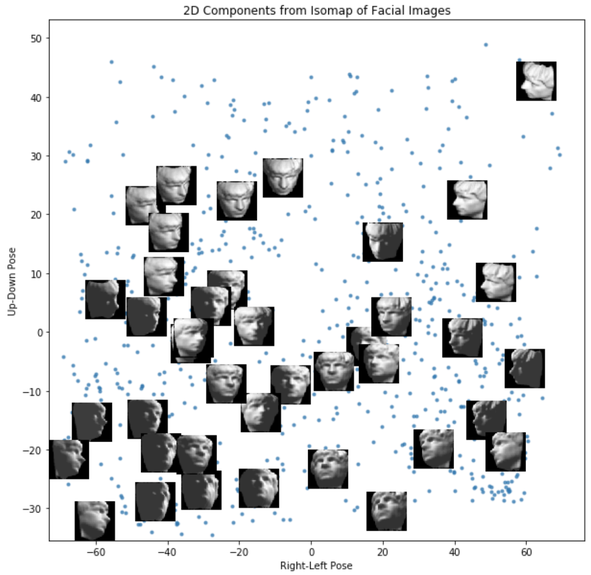


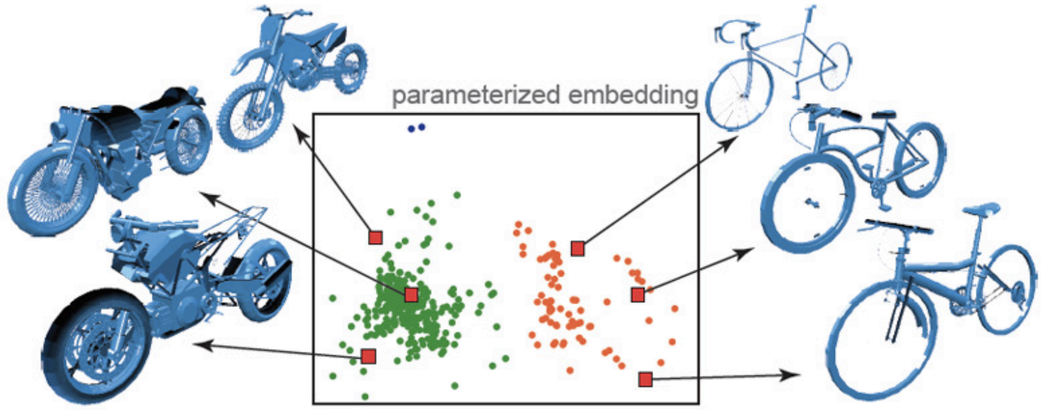
[Tenenbaum et al., Science, 2000]



[Yang et al., TOG, 2011]

Face Manifold





[Averkiou et al., Eurographics, 2014]

Example of Nonlinear Manifold: Faces

$\mathbf{x}_1$

$\mathbf{x}_2$

Example of Nonlinear Manifold: Faces

$\mathbf{x}_1$

$\frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2)$

$\mathbf{x}_2$

Example of Nonlinear Manifold: Faces

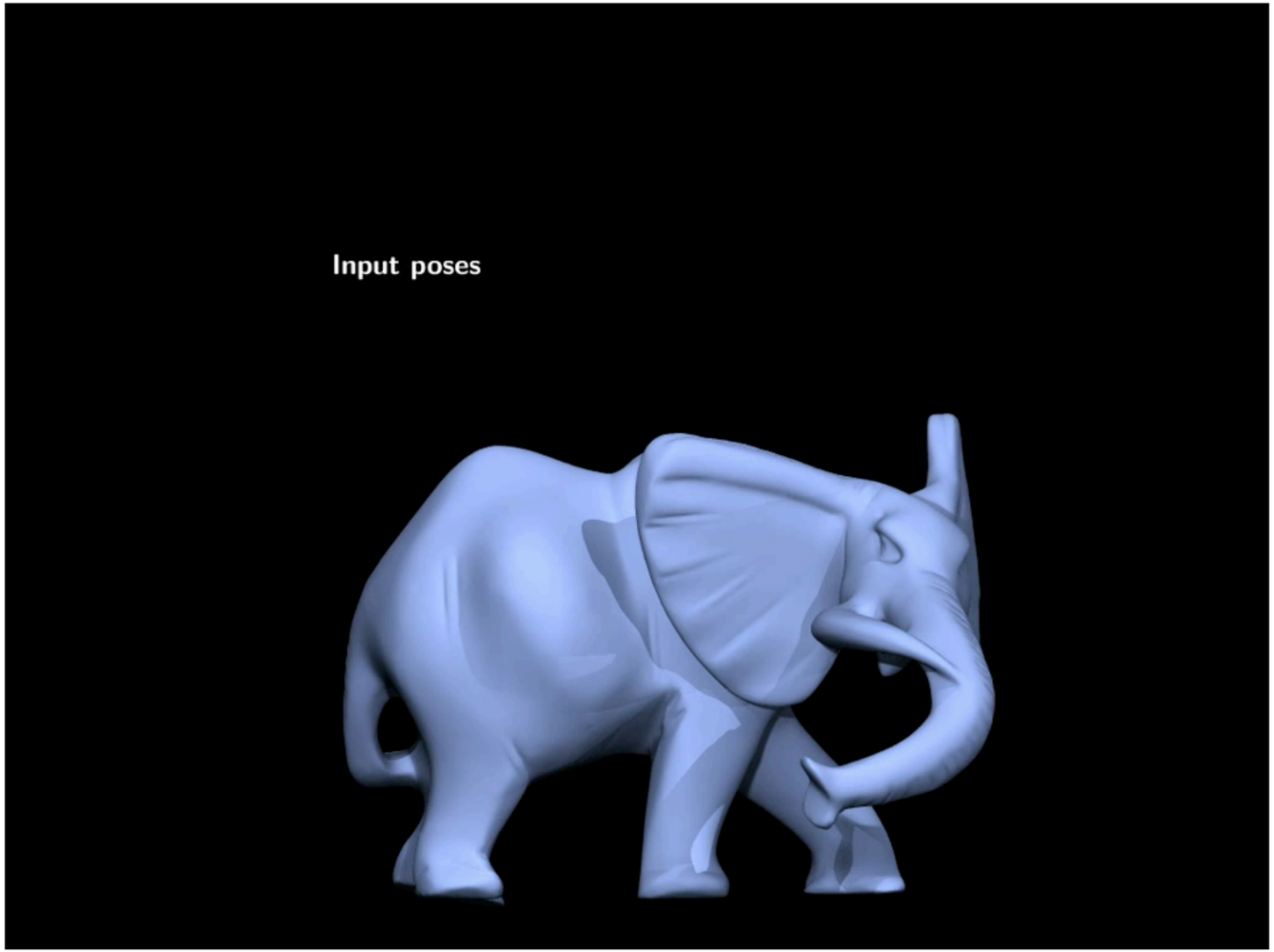
X

$\mathbf{x}_1$

$\frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2)$

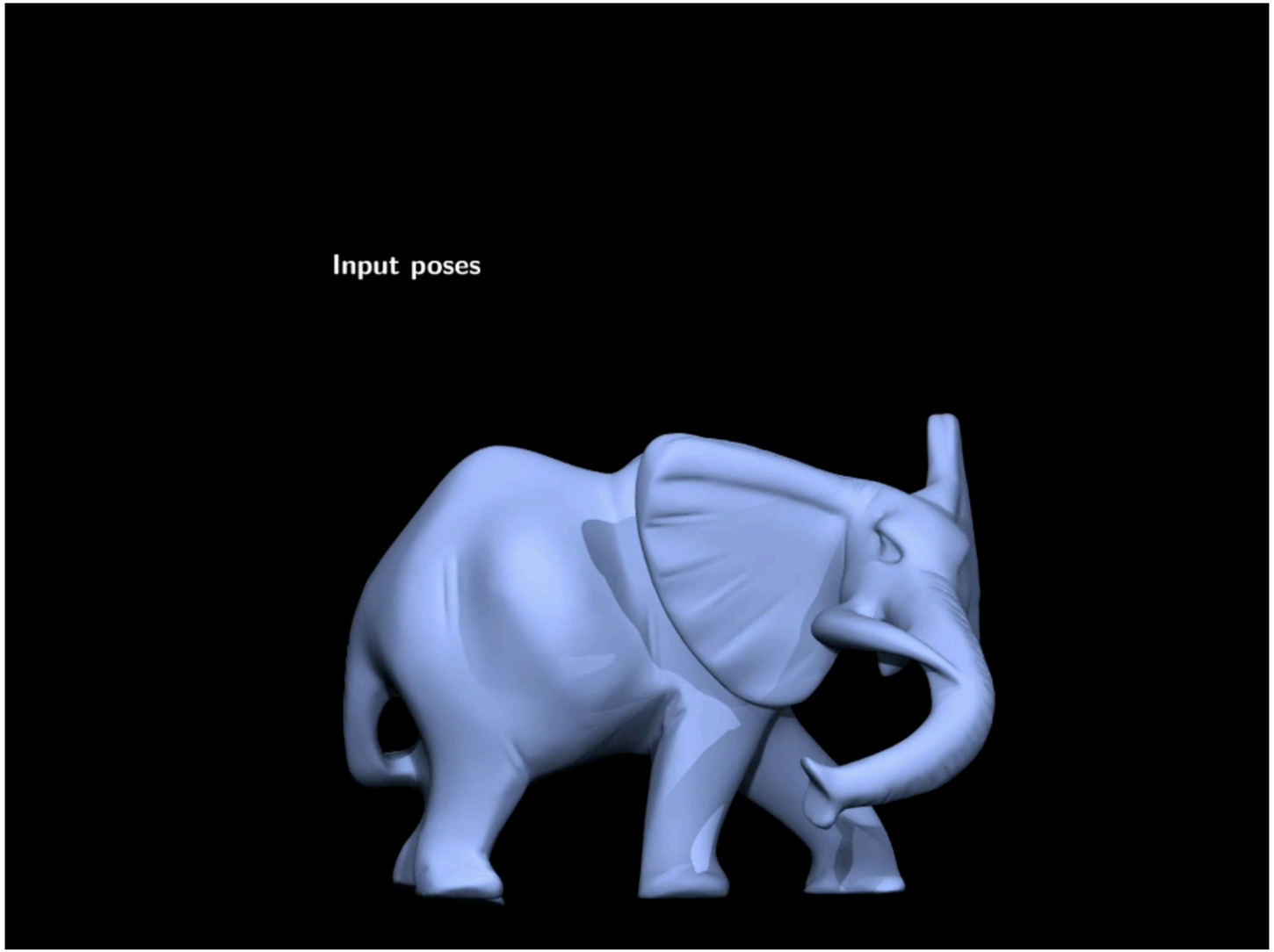
$\mathbf{x}_2$

Morphing (Interpolation in Shape Space)



[Kilian et al., Siggraph, 2007]

Morphing (Interpolation in Shape Space)



[Kilian et al., Siggraph, 2007]

Moving Along Learned Face Manifold

Trajectory along the “male” dimension

[Lample et. al. Fader Networks, NIPS 2017]

Moving Along Learned Face Manifold

Trajectory along the “male” dimension

Trajectory along the “young” dimension

[Lample et. al. Fader Networks, NIPS 2017]

Notations: Vectors and Matrices

Notations: Vectors and Matrices

vector $\mathbf{x}$

Notations: Vectors and Matrices

vector $\mathbf{x}$

matrix $\mathbf{A}_{m \times n} = [\mathbf{a}_1 \dots \mathbf{a}_n]$

Notations: Vectors and Matrices

vector	$\mathbf{x}$
matrix	$\mathbf{A}_{m \times n} = [\mathbf{a}_1 \dots \mathbf{a}_n]$
linear equation	$\mathbf{A}\mathbf{x} = \mathbf{b}$

Notations: Vectors and Matrices

vector	$\mathbf{x}$
matrix	$\mathbf{A}_{m \times n} = [\mathbf{a}_1 \dots \mathbf{a}_n]$
linear equation	$\mathbf{A}\mathbf{x} = \mathbf{b}$
inner prod.	$< \mathbf{x}, \mathbf{y} >= \mathbf{x}^T \mathbf{y}$

Notations: Vectors and Matrices

vector $\mathbf{x}$

matrix $\mathbf{A}_{m\times n} = [\mathbf{a}_1 \dots \mathbf{a}_n]$

linear equation $\mathbf{A}\mathbf{x} = \mathbf{b}$

inner prod. $< \mathbf{x}, \mathbf{y} >= \mathbf{x}^T\mathbf{y}$

$\|\mathbf{x}\| = \sqrt{\mathbf{x}^T\mathbf{x}}$

$\mathbf{x}^T\mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\|\cos(\theta)$

Notations: Vectors and Matrices

- linear **independence**; **rank** of a matrix
- **span** of a matrix

vector	$\mathbf{x}$
matrix	$\mathbf{A}_{m\times n} = [\mathbf{a}_1 \dots \mathbf{a}_n]$
linear equation	$\mathbf{A}\mathbf{x} = \mathbf{b}$
inner prod.	$< \mathbf{x}, \mathbf{y} >= \mathbf{x}^T\mathbf{y}$

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^T\mathbf{x}}$$

$$\mathbf{x}^T\mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\|\cos(\theta)$$

Notations: Vectors and Matrices (cont.)

Notations: Vectors and Matrices (cont.)

$$\|\mathbf{x}\|_p = \left(|x_1|^p + |x_2|^p + \dots\right)^{1/p}$$

Notations: Vectors and Matrices (cont.)

$$\|\mathbf{x}\|_p = \left(|x_1|^p + |x_2|^p + \dots\right)^{1/p}$$

$$\|\mathbf{x}\|_p = \max\{|x_1|, |x_2|, \dots\} \qquad p = \infty$$

Notations: Vectors and Matrices (cont.)

$$\|\mathbf{x}\|_p = \left(|x_1|^p + |x_2|^p + \dots\right)^{1/p}$$

$$\|\mathbf{x}\|_p = \max\{|x_1|, |x_2|, \dots\} \qquad p = \infty$$

$$L_1, L_2, L_p, L_\infty$$

Notations: Vectors and Matrices (cont.)

$$\|\mathbf{x}\|_p = \left(|x_1|^p + |x_2|^p + \dots\right)^{1/p}$$

$$L_1, L_2, L_p, L_\infty$$

$$\|\mathbf{x}\|_p = \max\{|x_1|, |x_2|, \dots\} \qquad p = \infty$$

range $\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\}$

Notations: Vectors and Matrices (cont.)

$$\|\mathbf{x}\|_p = \left(|x_1|^p + |x_2|^p + \dots\right)^{1/p}$$

$$\|\mathbf{x}\|_p = \max\{|x_1|, |x_2|, \dots\} \qquad p = \infty$$

$$L_1, L_2, L_p, L_\infty$$

range $\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\}$

null space $\mathcal{N}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = 0\}$

Eigenvectors and Eigenvalues

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

Eigenvectors and Eigenvalues

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\mathbf{A}\mathbf{e}_i = \lambda_i \mathbf{e}_i$$

Eigenvectors and Eigenvalues

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\mathbf{A}\mathbf{e}_i = \lambda_i \mathbf{e}_i$$

$$\mathbf{T} = [\mathbf{v}_1\ \mathbf{v}_2 \dots]$$

$$\mathbf{T}^{-1} A \mathbf{T} = \mathrm{diag}(\lambda_1, \lambda_2, \dots)$$

Eigenvectors and Eigenvalues

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

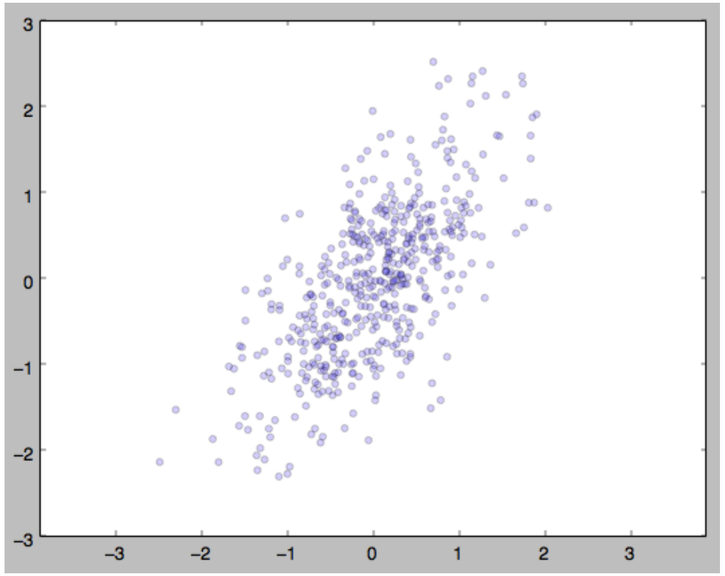
$$\mathbf{A}\mathbf{e}_i = \lambda_i \mathbf{e}_i$$

$$\mathbf{T} = [\mathbf{v}_1\ \mathbf{v}_2 \dots]$$

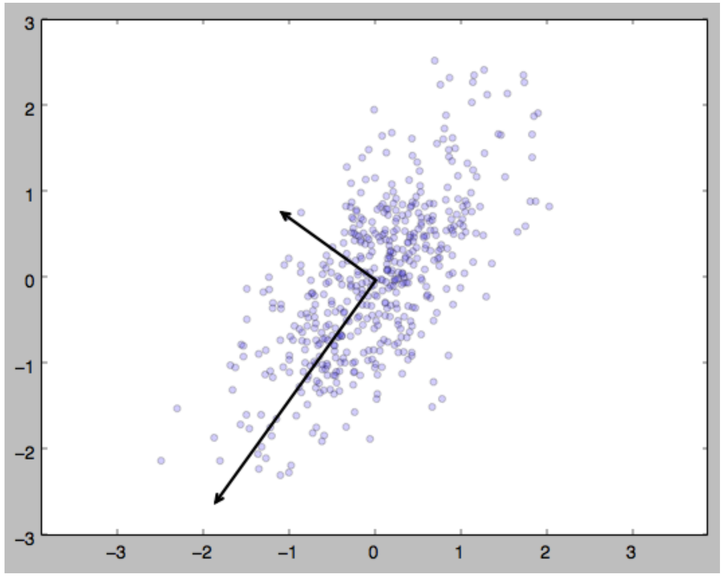
$$\mathbf{T}^{-1} A \mathbf{T} = \mathrm{diag}(\lambda_1, \lambda_2, \dots)$$

- All eigenvalues of symmetric matrices are real.
- Any real symmetric nxn matrix has a set of **n** mutually orthogonal eigenvectors.

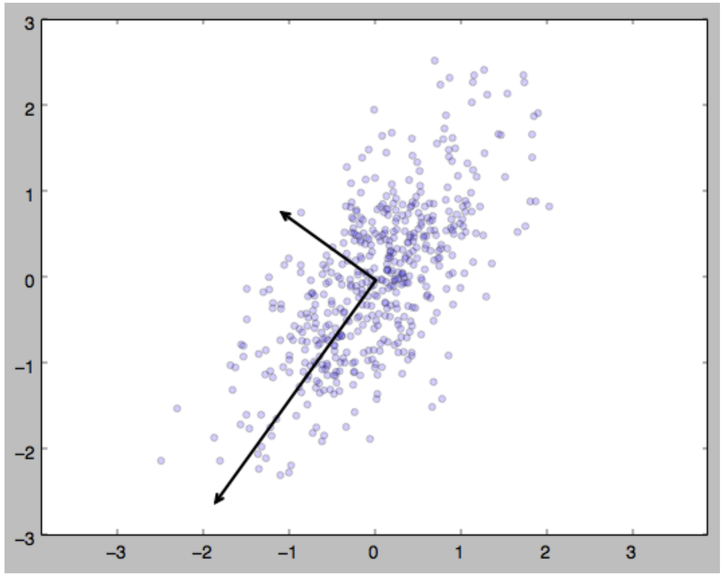
Code Example



Code Example

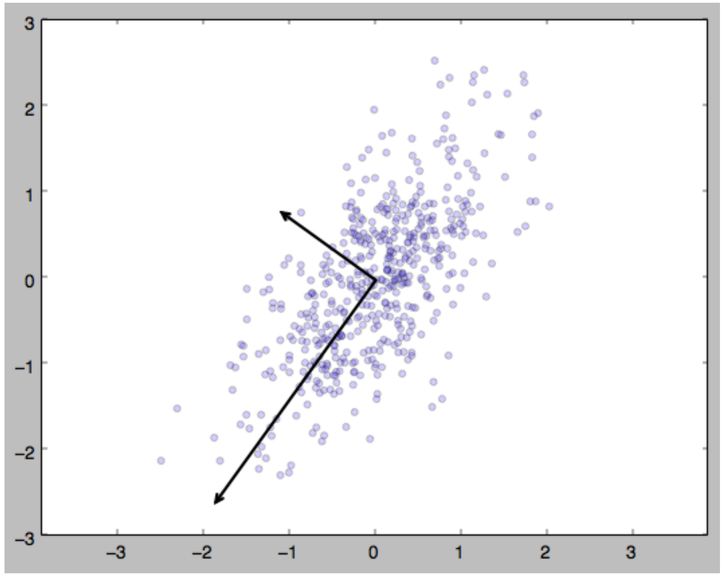


Code Example



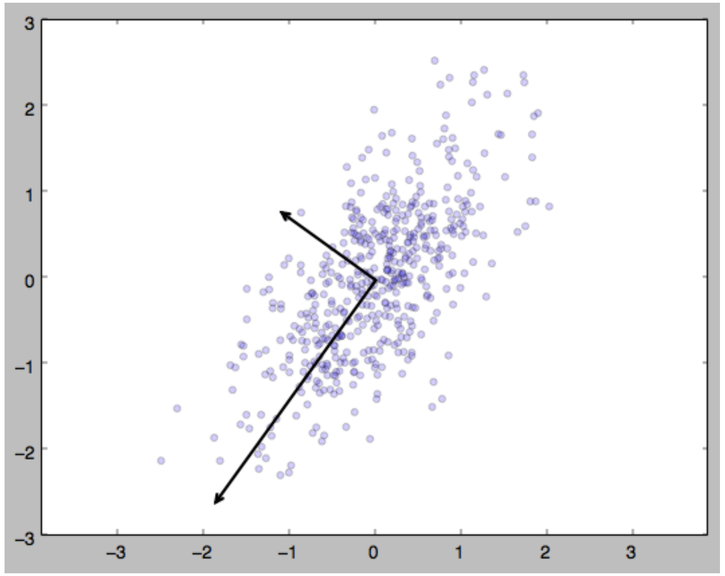

```
rng = np.random.RandomState(10)
X = np.dot(rng.rand(2, 2), rng.randn(2, 500)).T

mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Code Example



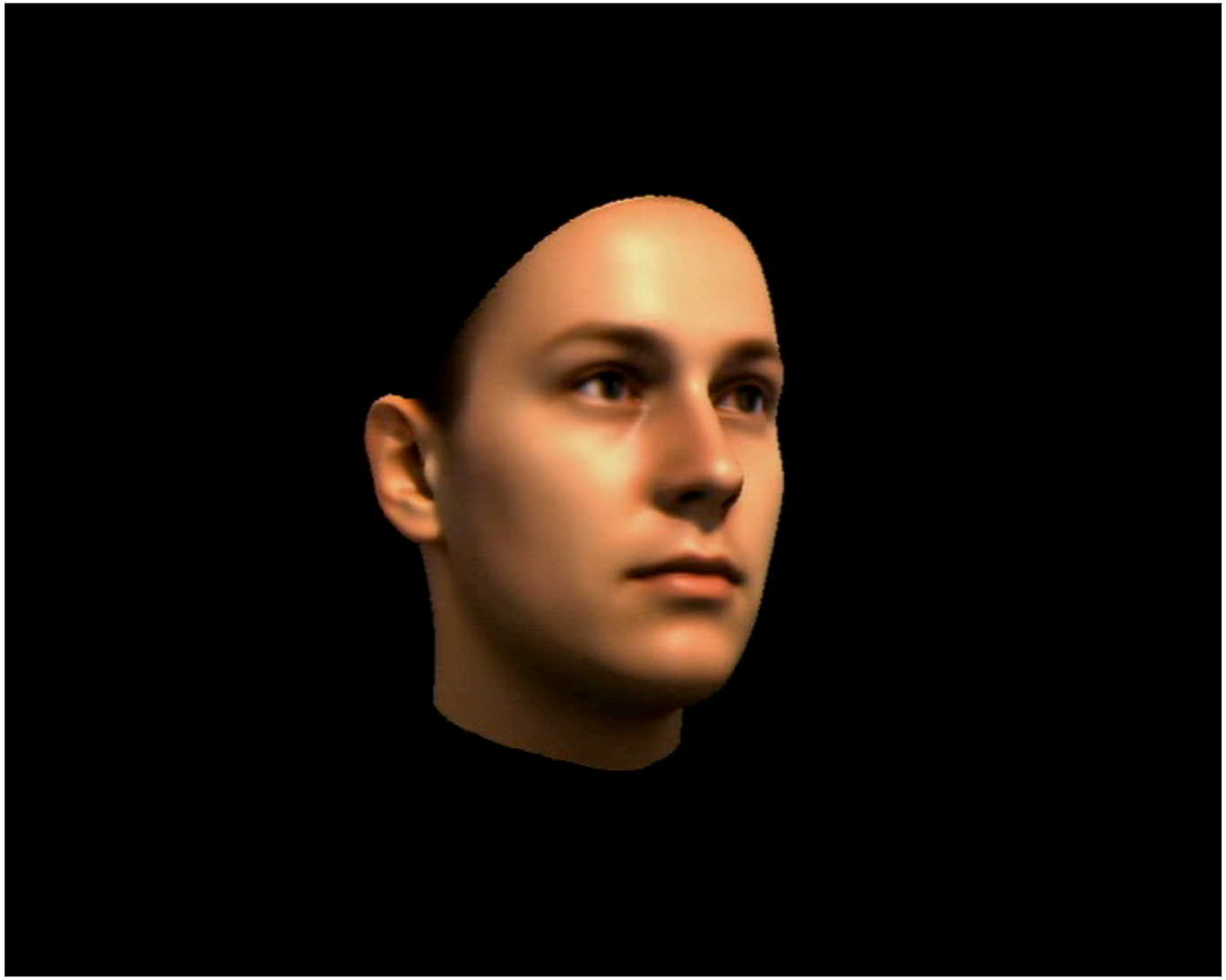

```
rng = np.random.RandomState(10)
X = np.dot(rng.rand(2, 2), rng.randn(2, 500)).T

mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Code Example

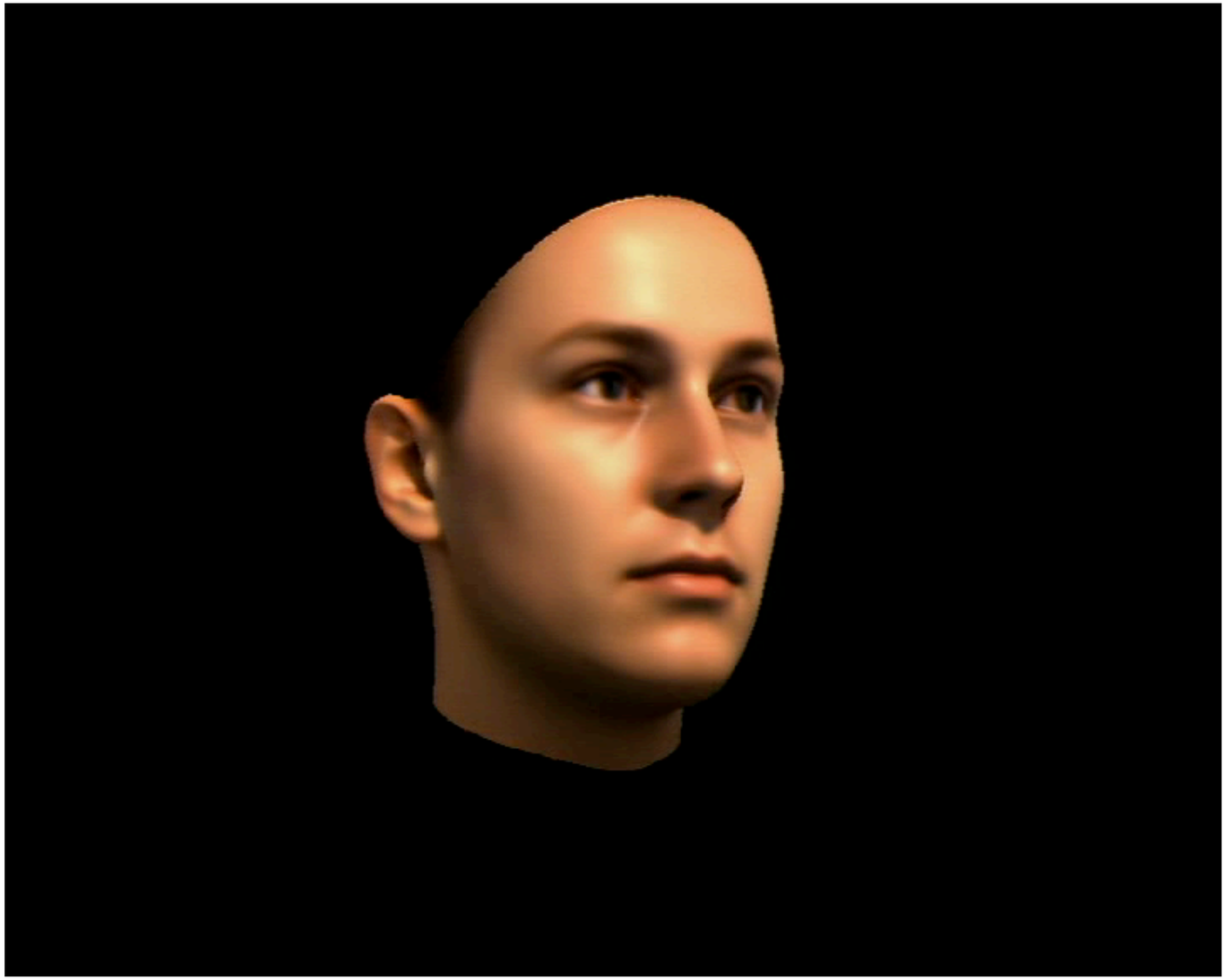



```
rng = np.random.RandomState(10)
X = np.dot(rng.rand(2, 2), rng.randn(2, 500)).T

mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Morphable Faces



Morphable Faces



Singular Value Decomposition (SVD)

- Very useful for matrix manipulation.
- Used for robust numerical computation.

Singular Value Decomposition (SVD)

- Very useful for matrix manipulation.
- Used for robust numerical computation.

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

Singular Value Decomposition (SVD)

- Very useful for matrix manipulation.
- Used for robust numerical computation.

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$

rotation

Singular Value Decomposition (SVD)

- Very useful for matrix manipulation.
- Used for robust numerical computation.

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

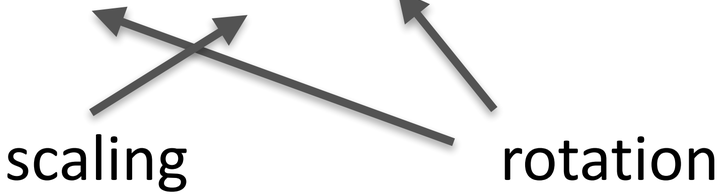
scaling

rotation

Singular Value Decomposition (SVD)

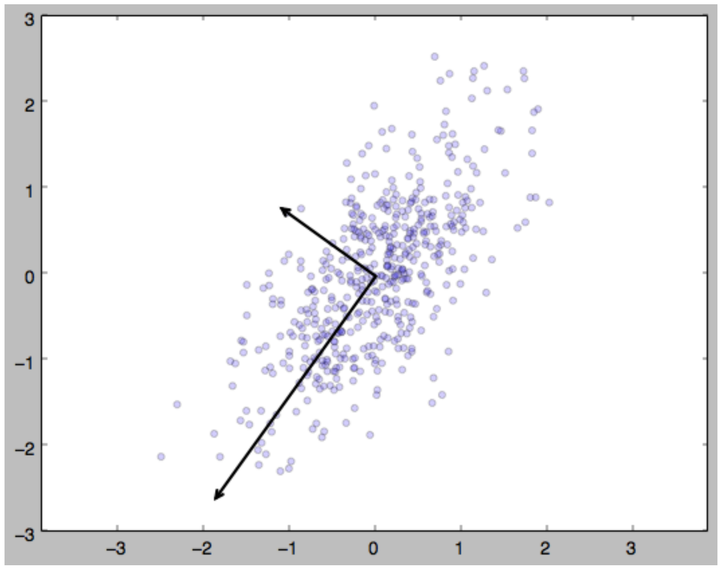
- Very useful for matrix manipulation.
- Used for robust numerical computation.

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$



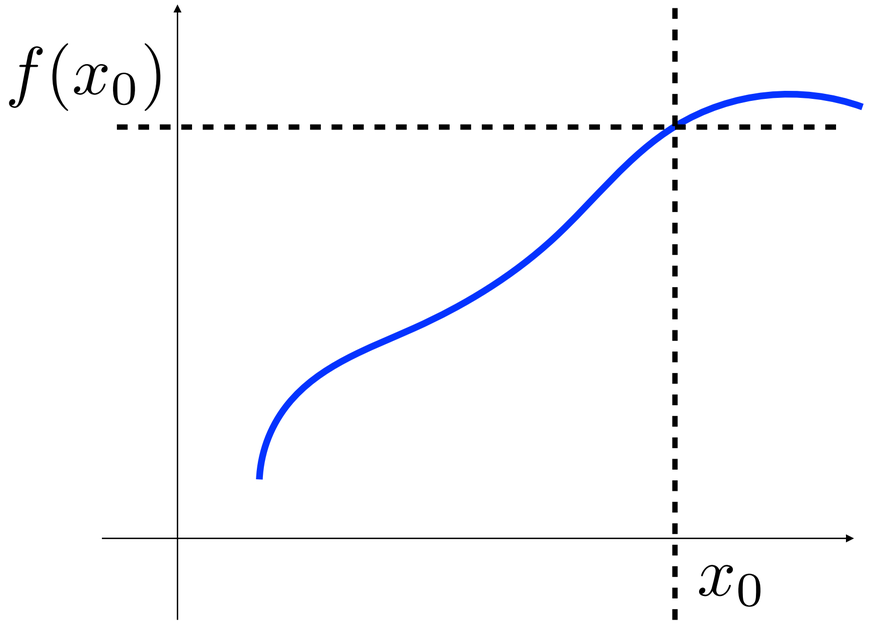
$$\mathbf{A} = \mathbf{A}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$$

Code Example

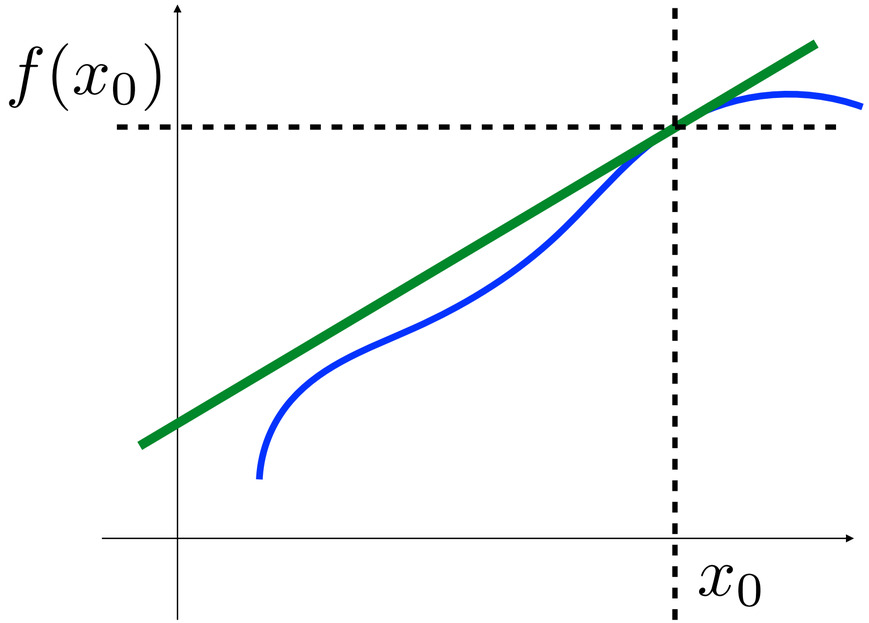



```
mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
matU, sigma, matV = np.linalg.svd(cov_mat)
```

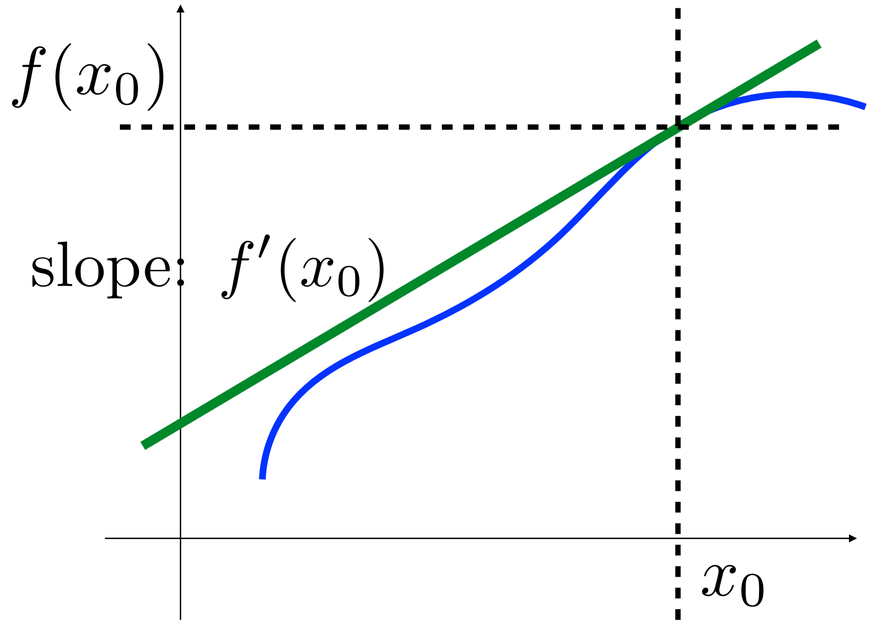
Differentiation (chain rule recap)



Differentiation (chain rule recap)

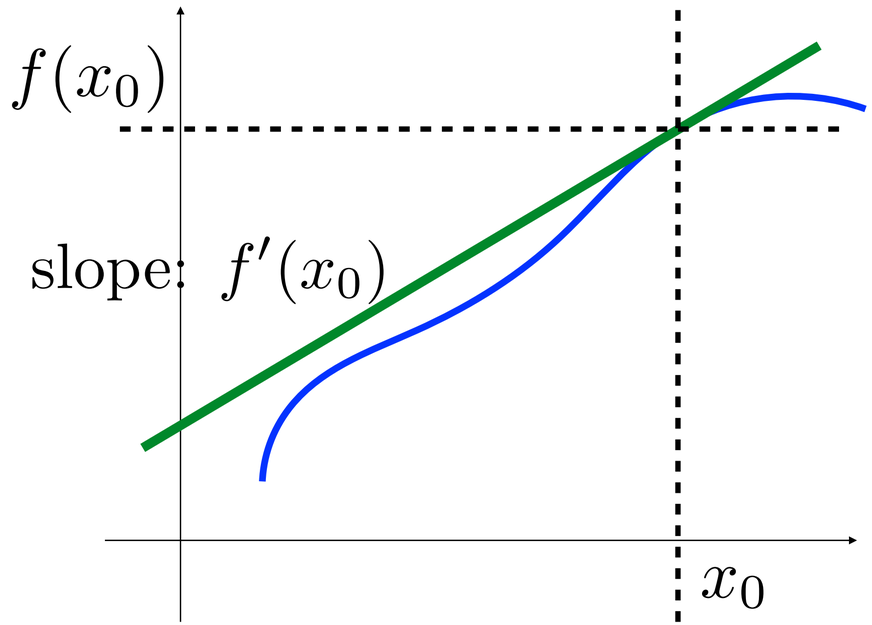


Differentiation (chain rule recap)



Differentiation (chain rule recap)

$$z = f \circ g(x) = f(g(x))$$

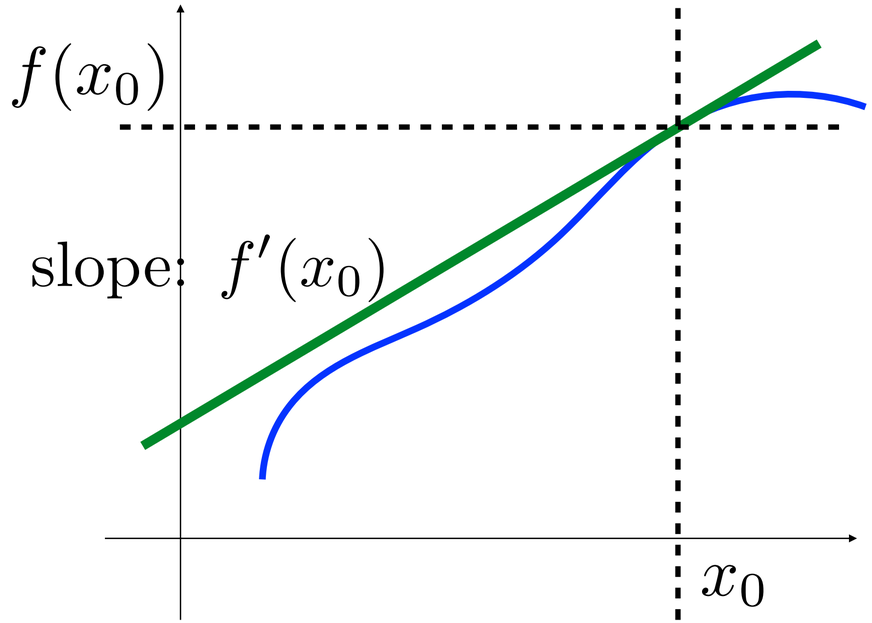


Differentiation (chain rule recap)

$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$



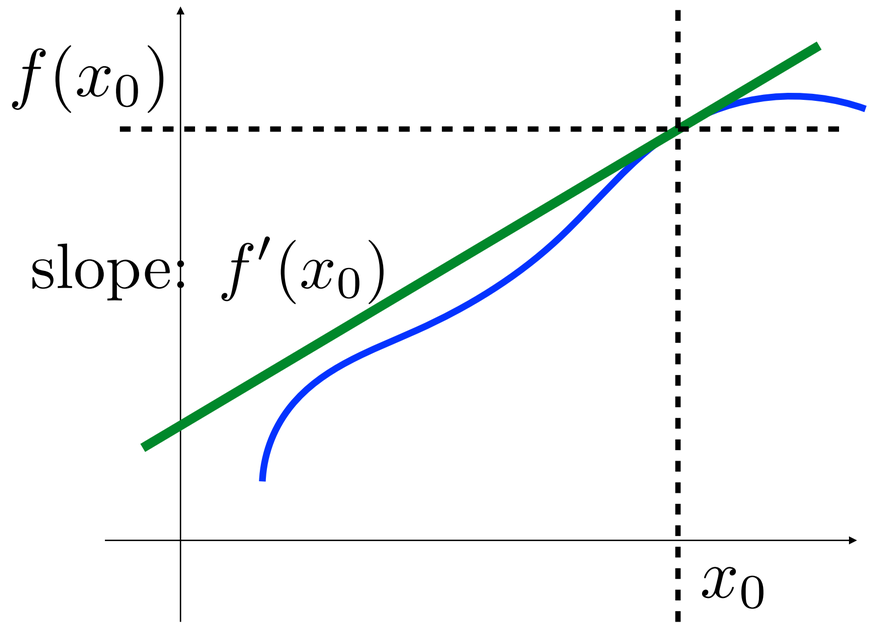
Differentiation (chain rule recap)

$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$



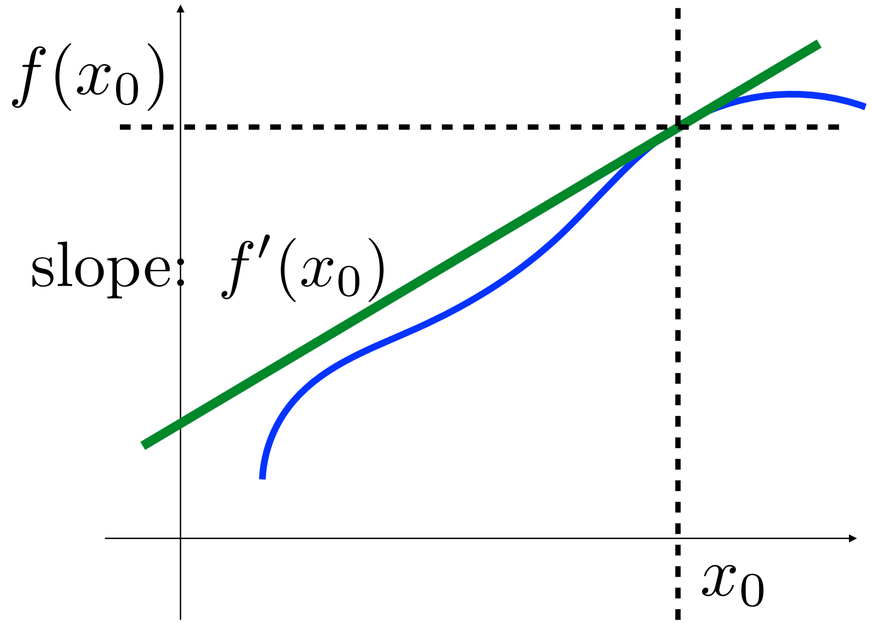
Differentiation (chain rule recap)

$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x)$$



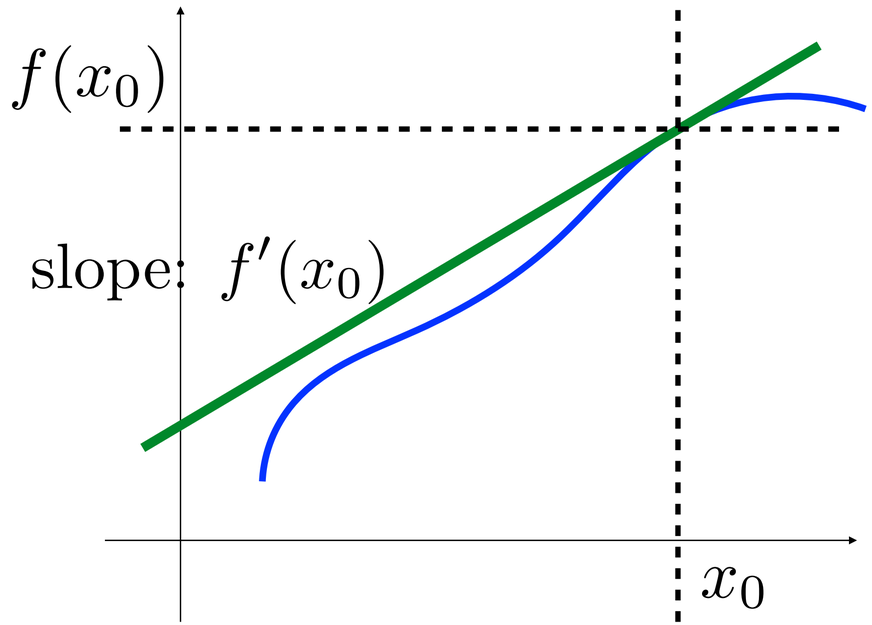
Differentiation (chain rule recap)

$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$



Differentiation (chain rule recap)

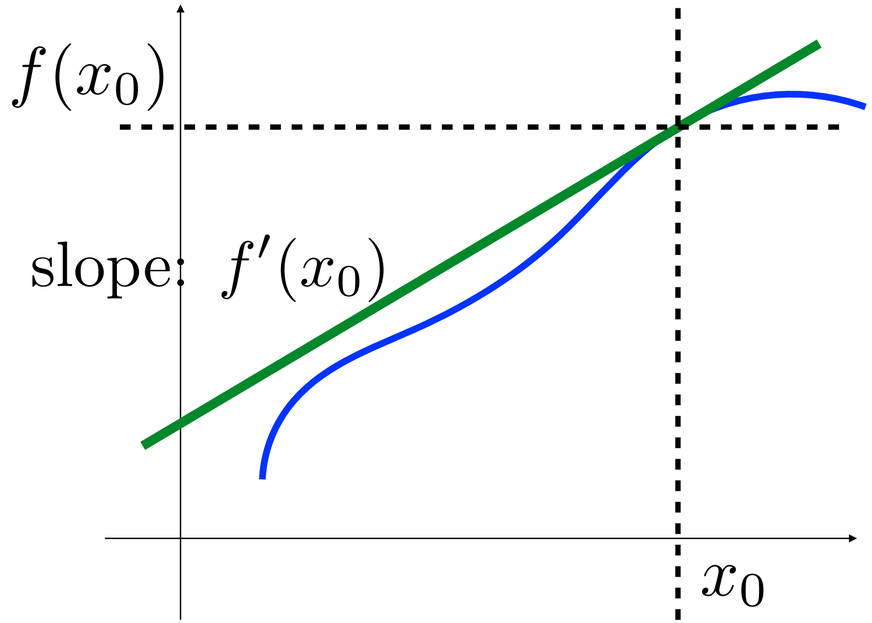
$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

$z = \sin(5x)$



Differentiation (chain rule recap)

$$z = f \circ g(x) = f(g(x))$$

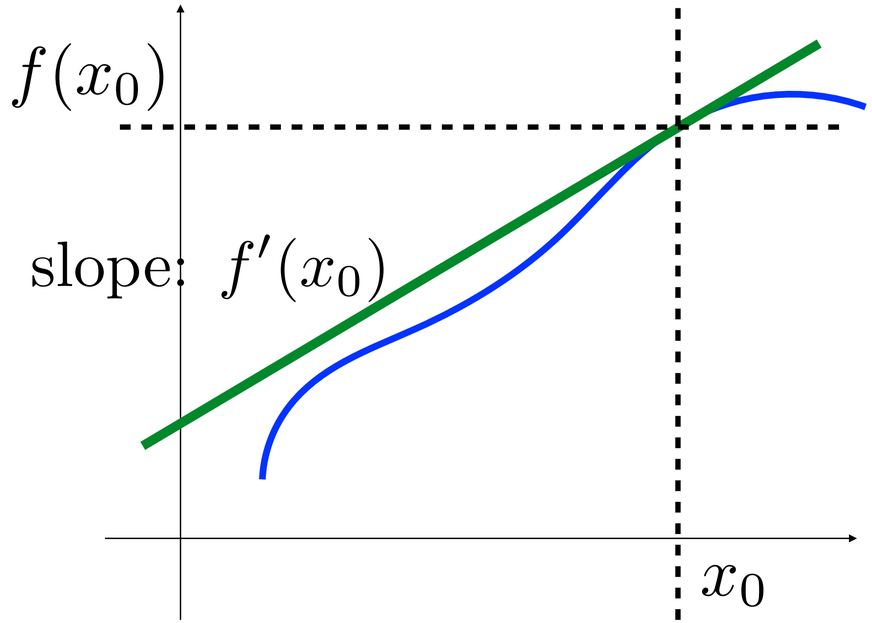
$$z = f(y)$$

$$y = g(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

$$z = \sin(5x)$$

$$= \frac{d\sin(5x)}{d(5x)} \frac{d(5x)}{dx}$$



Differentiation (chain rule recap)

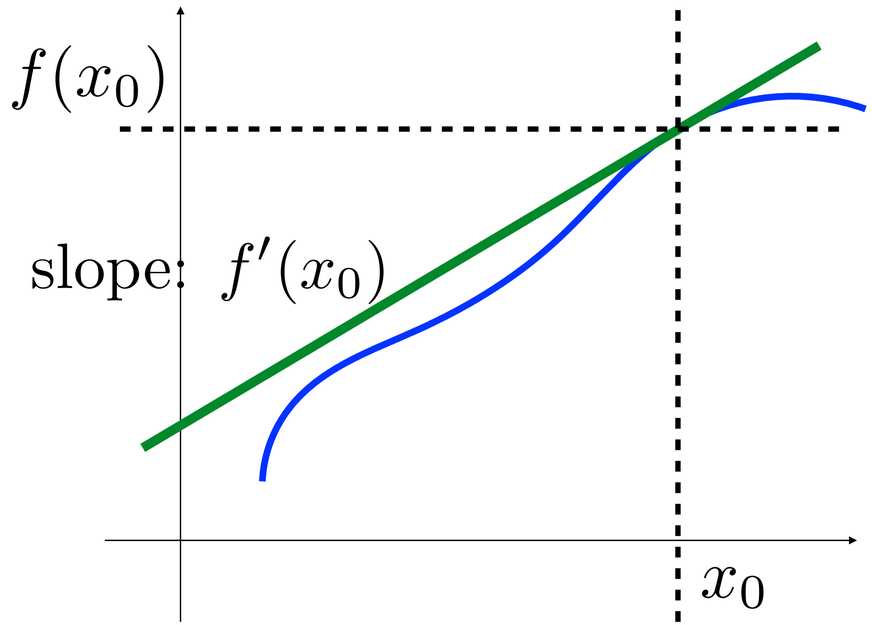
$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

$$\begin{aligned} z &= \sin(5x) \\ &= \frac{d\sin(5x)}{d(5x)} \frac{d(5x)}{dx} \\ &= 5\cos(5x) \end{aligned}$$



Derivative Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Derivative Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix}$$

Derivative Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} \qquad \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_j} = \begin{bmatrix} \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial x_j} \\ \vdots \\ \frac{\partial \mathbf{f}_m(\mathbf{x})}{\partial x_j} \end{bmatrix}$$

Derivative Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} \qquad \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_j} = \begin{bmatrix} \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial x_j} \\ \vdots \\ \frac{\partial \mathbf{f}_m(\mathbf{x})}{\partial x_j} \end{bmatrix}$$

$$\mathbf{L} = D\mathbf{f} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

Derivative Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} \qquad \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_j} = \begin{bmatrix} \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial x_j} \\ \vdots \\ \frac{\partial \mathbf{f}_m(\mathbf{x})}{\partial x_j} \end{bmatrix}_{m \times 1}$$

$$\mathbf{L} = D\mathbf{f} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

Derivative Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} \qquad \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_j} = \begin{bmatrix} \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial x_j} \\ \vdots \\ \frac{\partial \mathbf{f}_m(\mathbf{x})}{\partial x_j} \end{bmatrix}_{m \times 1}$$

$$\mathbf{L} = D\mathbf{f} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}_{m \times n}$$

Derivative Matrix

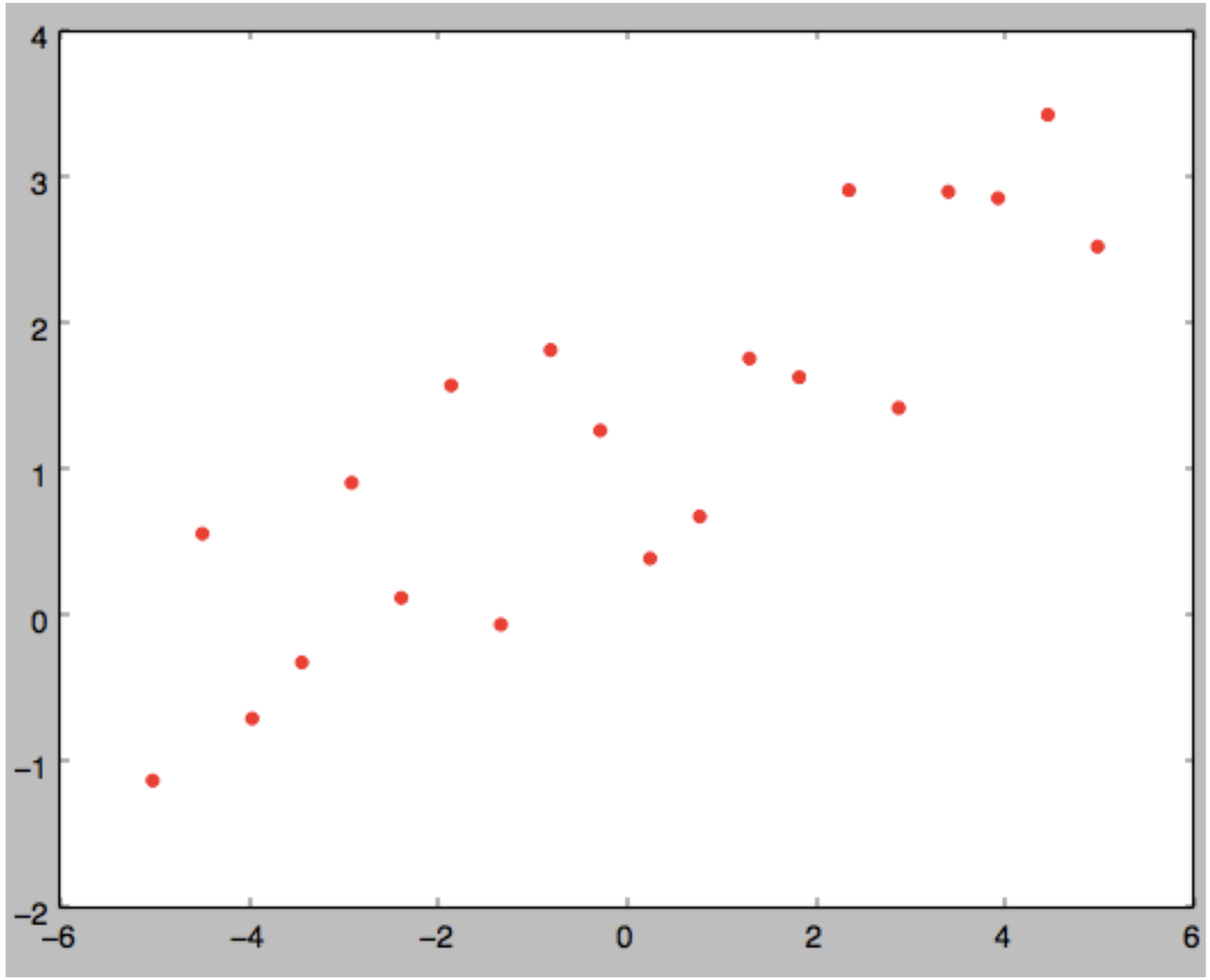
$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} \qquad \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_j} = \begin{bmatrix} \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial x_j} \\ \vdots \\ \frac{\partial \mathbf{f}_m(\mathbf{x})}{\partial x_j} \end{bmatrix}_{m \times 1}$$

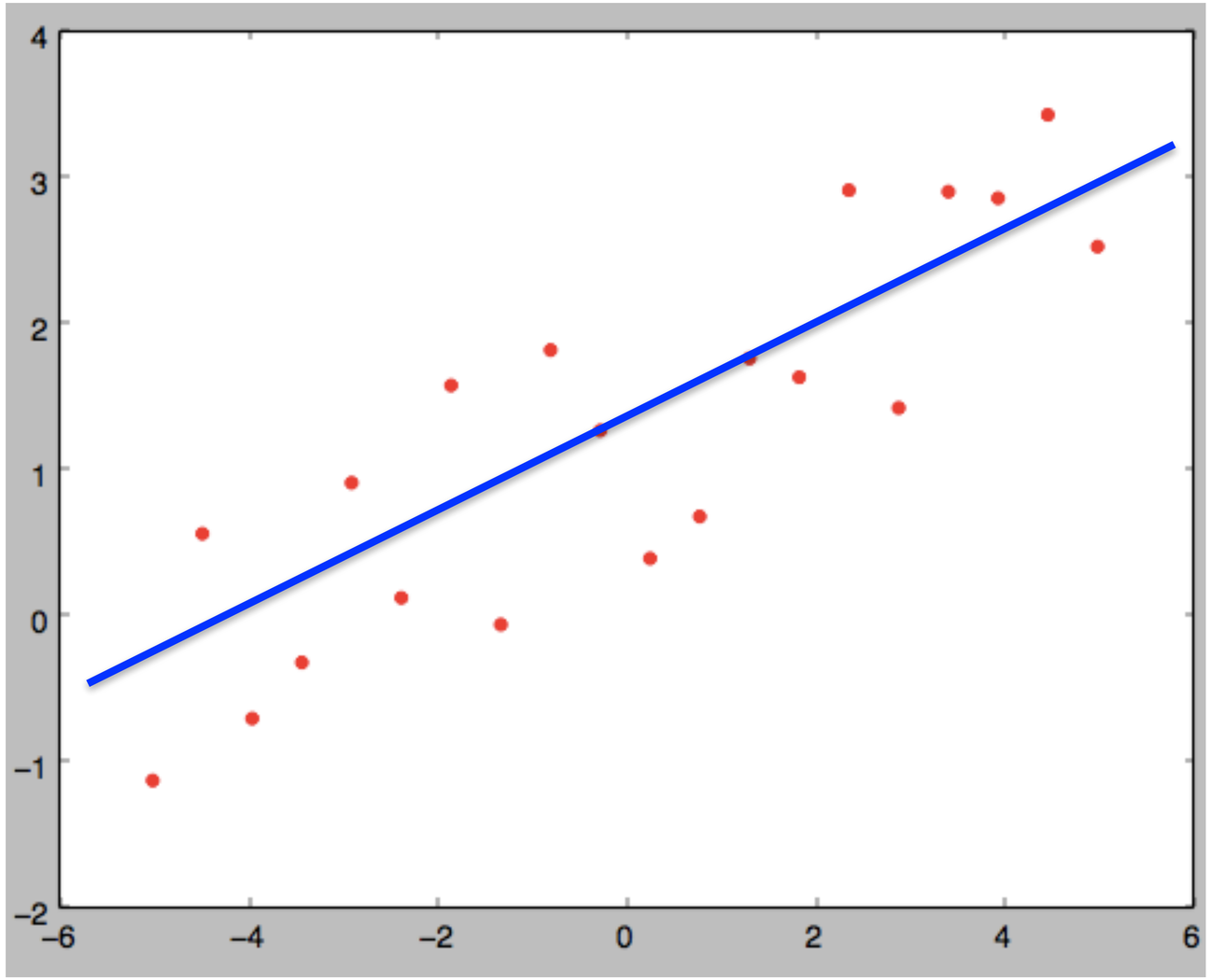
Jacobian matrix

$$\mathbf{L} = D\mathbf{f} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}_{m \times n}$$

Regression: Continuous Output



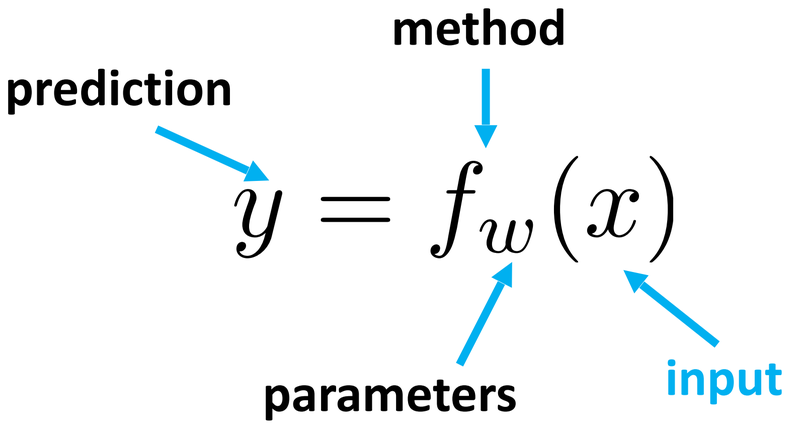
Regression: Continuous Output



Learning a Function

$$y = f_w(x)$$

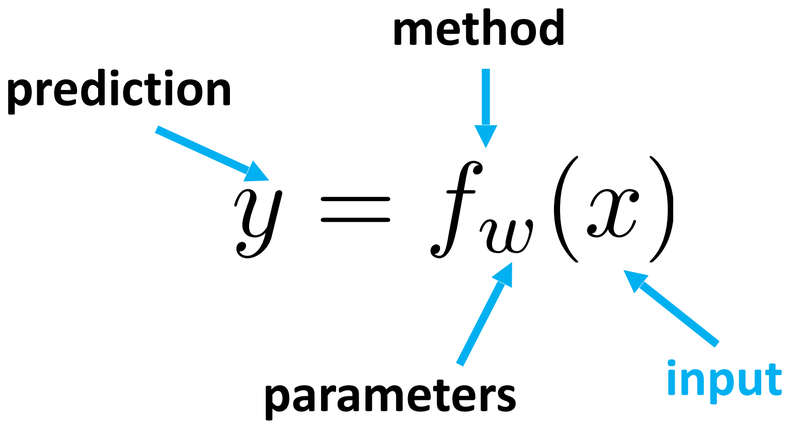
Learning a Function



Calculus $x \in \mathbb{R}$

Vector calculus $\mathbf{x} \in \mathbb{R}^d$

Learning a Function

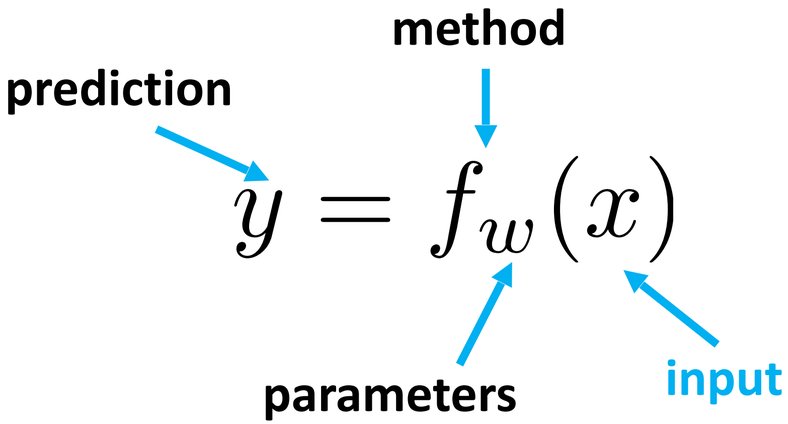


Calculus $x \in \mathbb{R}$

Vector calculus $\mathbf{x} \in \mathbb{R}^d$

Machine learning: can work also for discrete inputs, strings, images, meshes, animations, …

Learning a Function



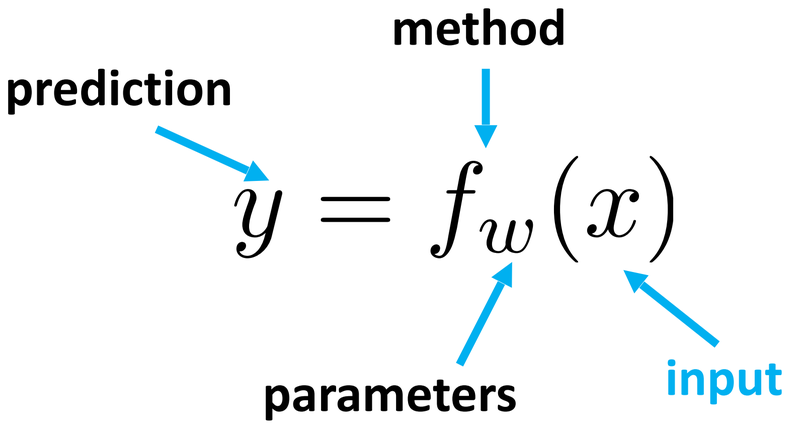
Calculus $x \in \mathbb{R}$

Vector calculus $\mathbf{x} \in \mathbb{R}^d$

Classification: $y \in \{0, 1\}$

Machine learning: can work also for discrete inputs, strings, images, meshes, animations, …

Learning a Function



Calculus $x \in \mathbb{R}$

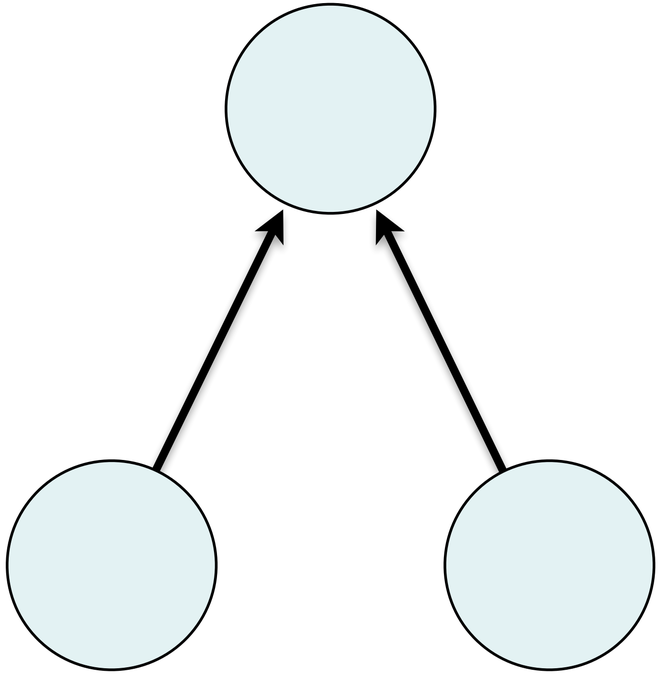
Vector calculus $\mathbf{x} \in \mathbb{R}^d$

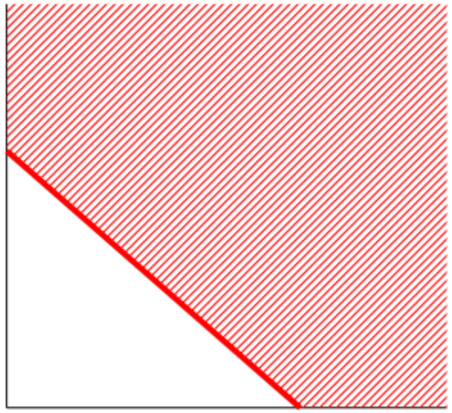
Classification: $y \in \{0, 1\}$

Regression: $y \in \mathbb{R}$

Machine learning: can work also for discrete inputs, strings, images, meshes, animations, …

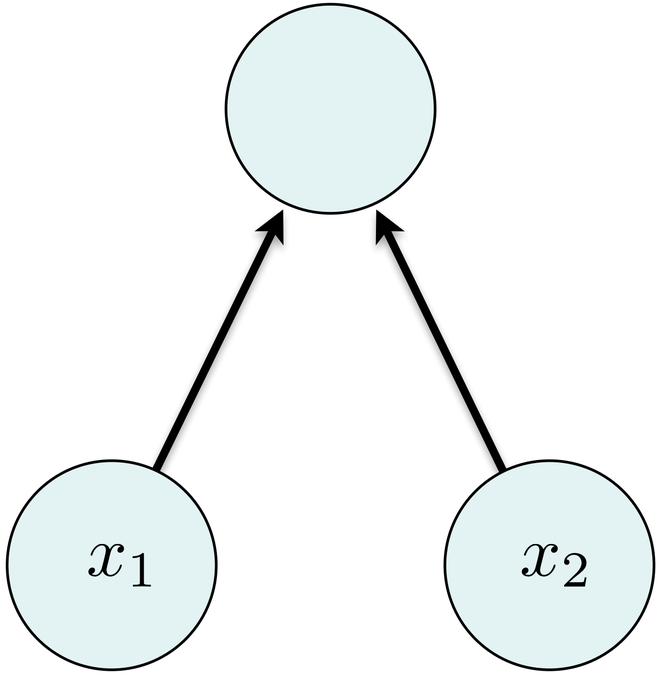
Learning a Simple Separator/Classifier

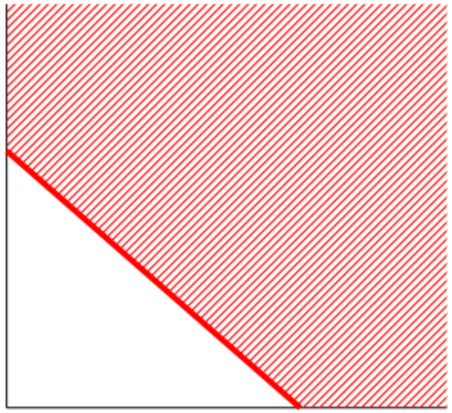




separating hyperplane

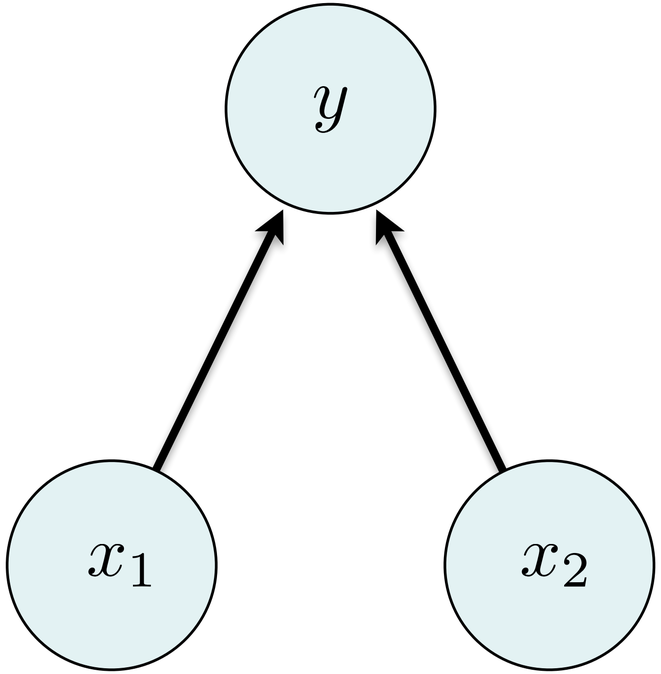
Learning a Simple Separator/Classifier

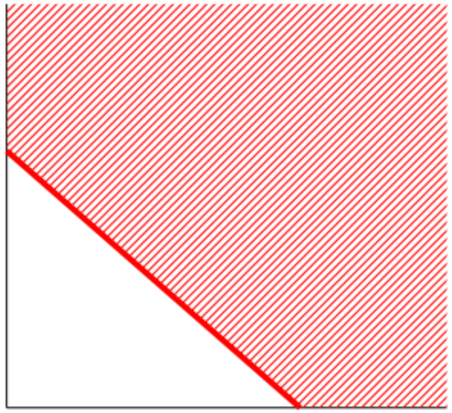




separating hyperplane

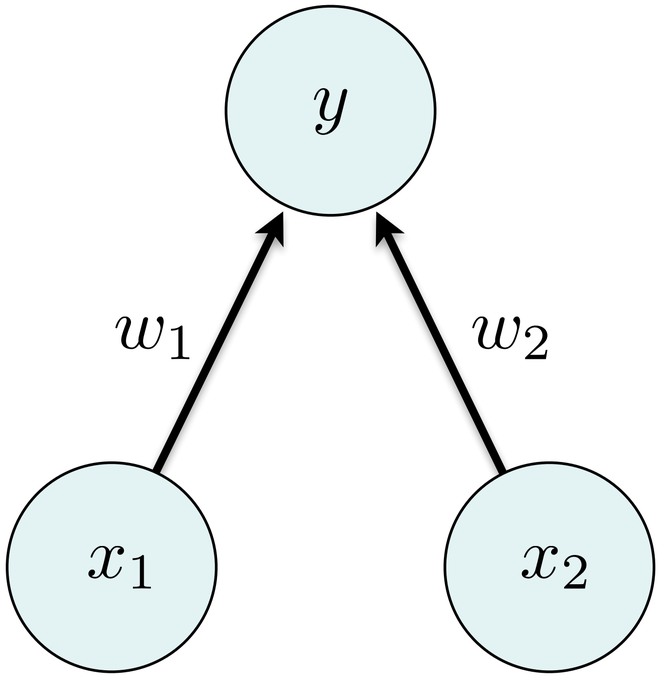
Learning a Simple Separator/Classifier

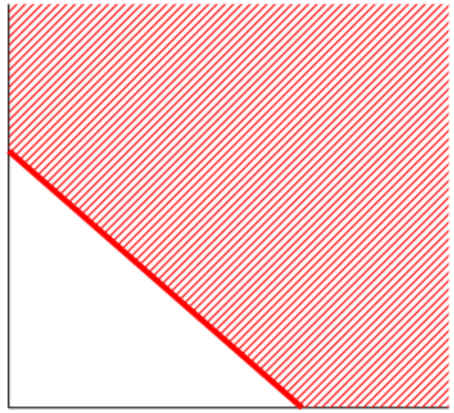




separating hyperplane

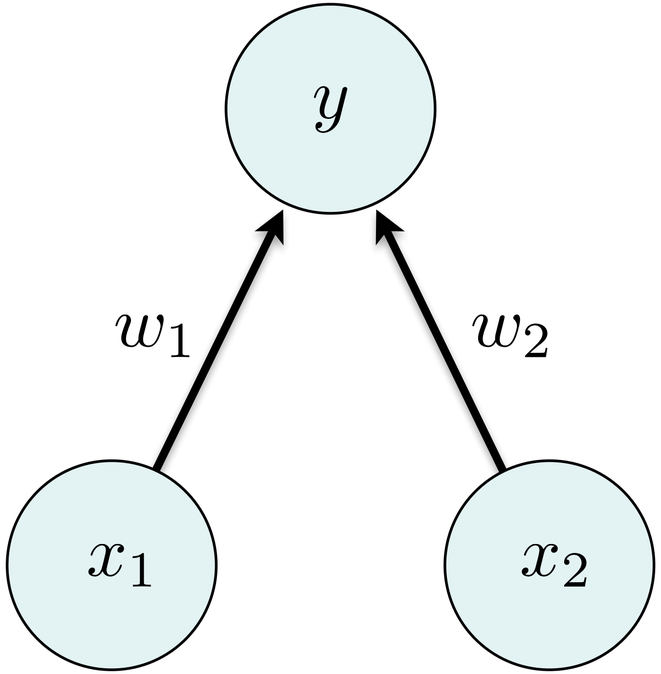
Learning a Simple Separator/Classifier

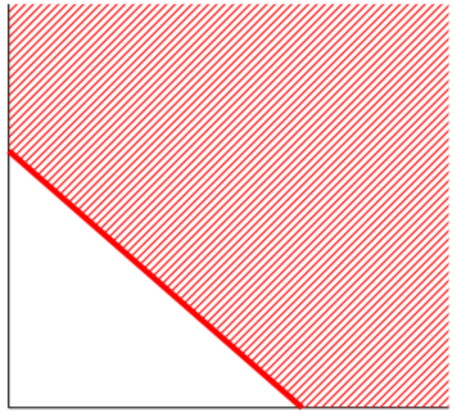




separating hyperplane

Learning a Simple Separator/Classifier

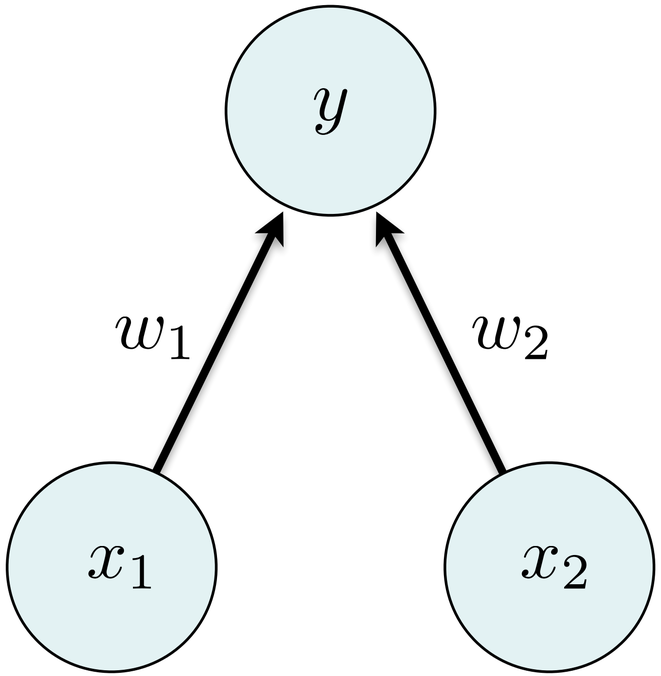


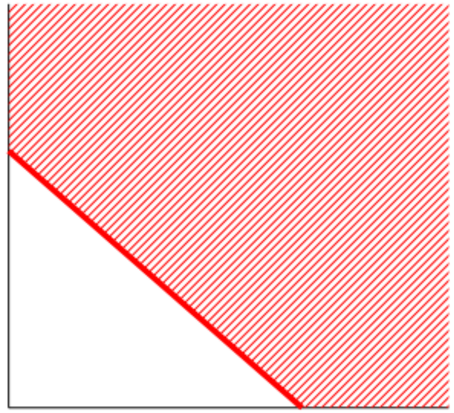


separating hyperplane

$$y = f(w_1 x_1 + w_2 x_2)$$

Learning a Simple Separator/Classifier

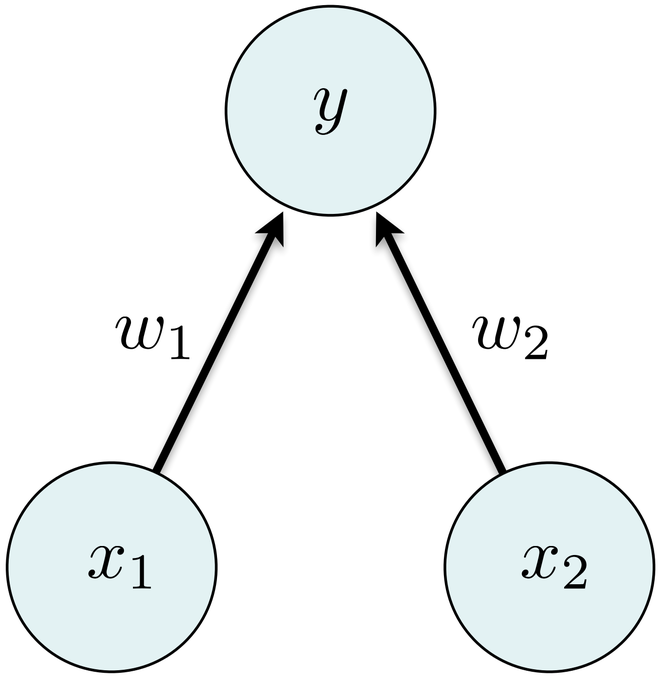


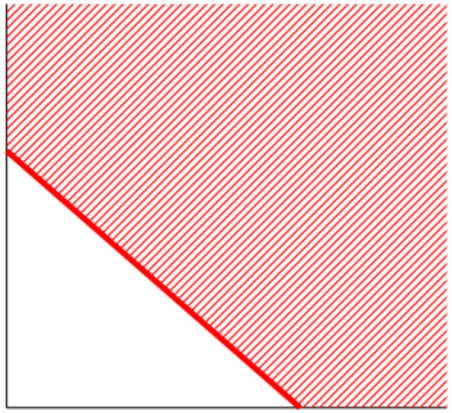


separating hyperplane

$$y = f(w_1 x_1 + w_2 x_2) = \mathcal{H}(w_1 x_1 + w_2 x_2)$$

Learning a Simple Separator/Classifier



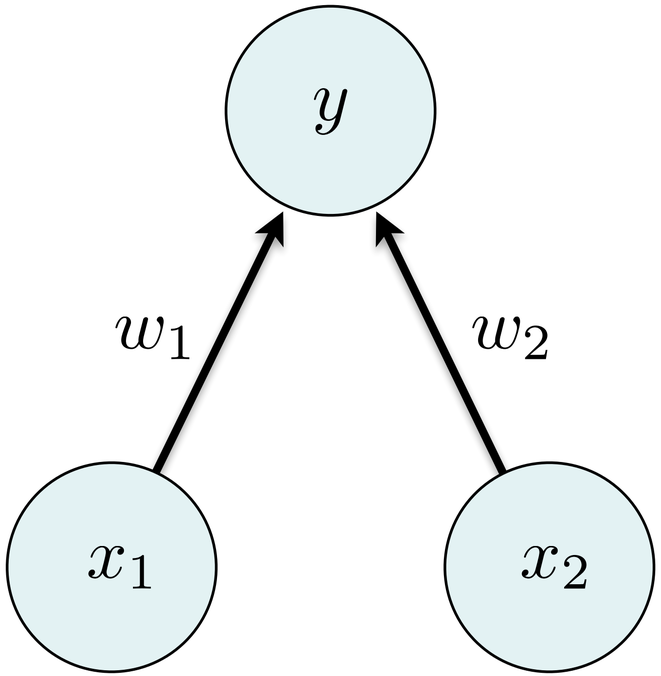


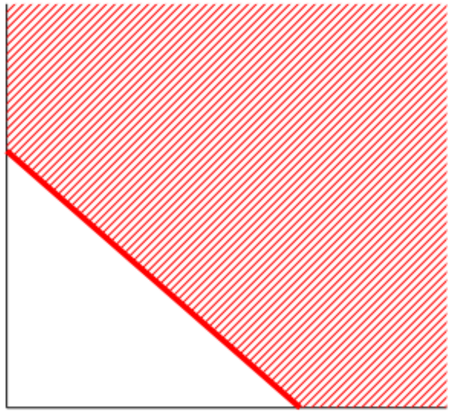
separating hyperplane

fixed non-linearity

$$y = f(w_1 x_1 + w_2 x_2) = \mathcal{H}(w_1 x_1 + w_2 x_2)$$

Learning a Simple Separator/Classifier





separating hyperplane

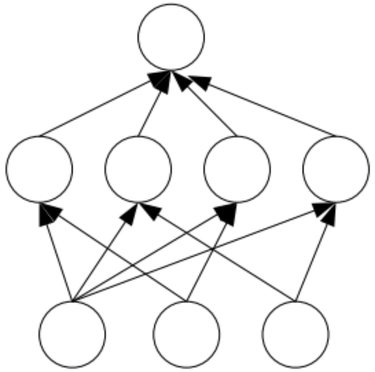
fixed non-linearity

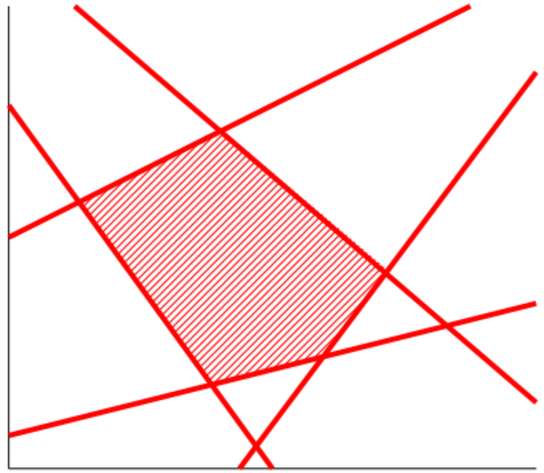
learned

$$y = f(w_1 x_1 + w_2 x_2) = \mathcal{H}(w_1 x_1 + w_2 x_2)$$

Combining Simple Functions/Classifiers

2 layers of trainable weights

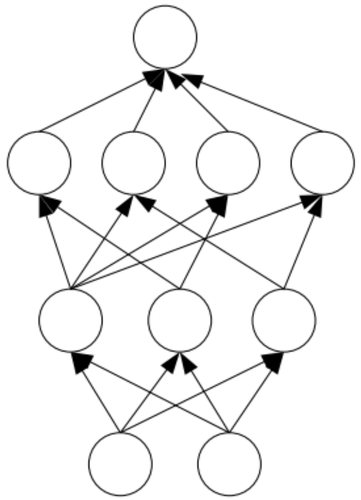


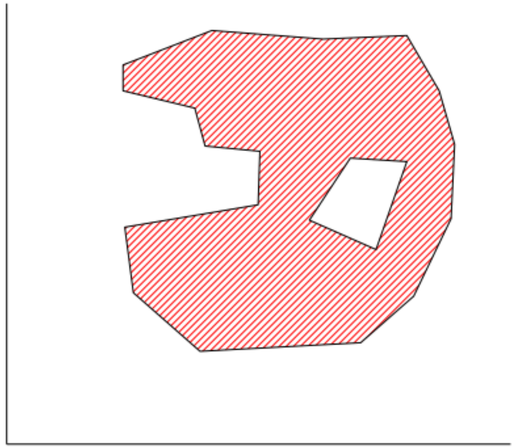


convex region

Combining Simple Functions/Classifiers

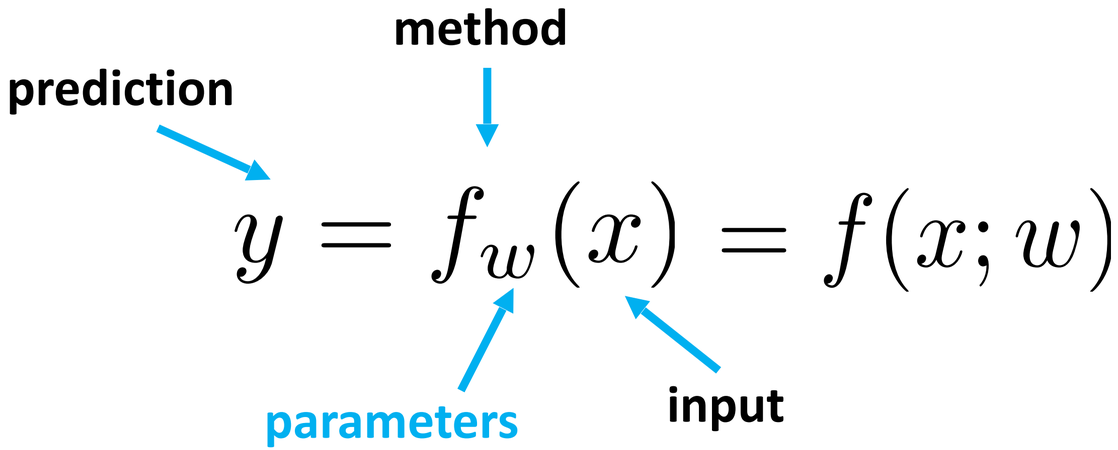
3 layers of trainable weights





complex polygons

Learning a Function: Modeling



$$w \in \mathbb{R}$$

$$\mathbf{w} \in \mathbb{R}^K$$

Regression

1. Least Squares fitting

2. Nonlinear error function and gradient descent

3. Perceptron training

Regression

1. **Least Squares fitting**

2. Nonlinear error function and gradient descent

3. Perceptron training

Assumption: Linear Function

$$y = f_{\mathbf{w}}(\mathbf{x}) = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

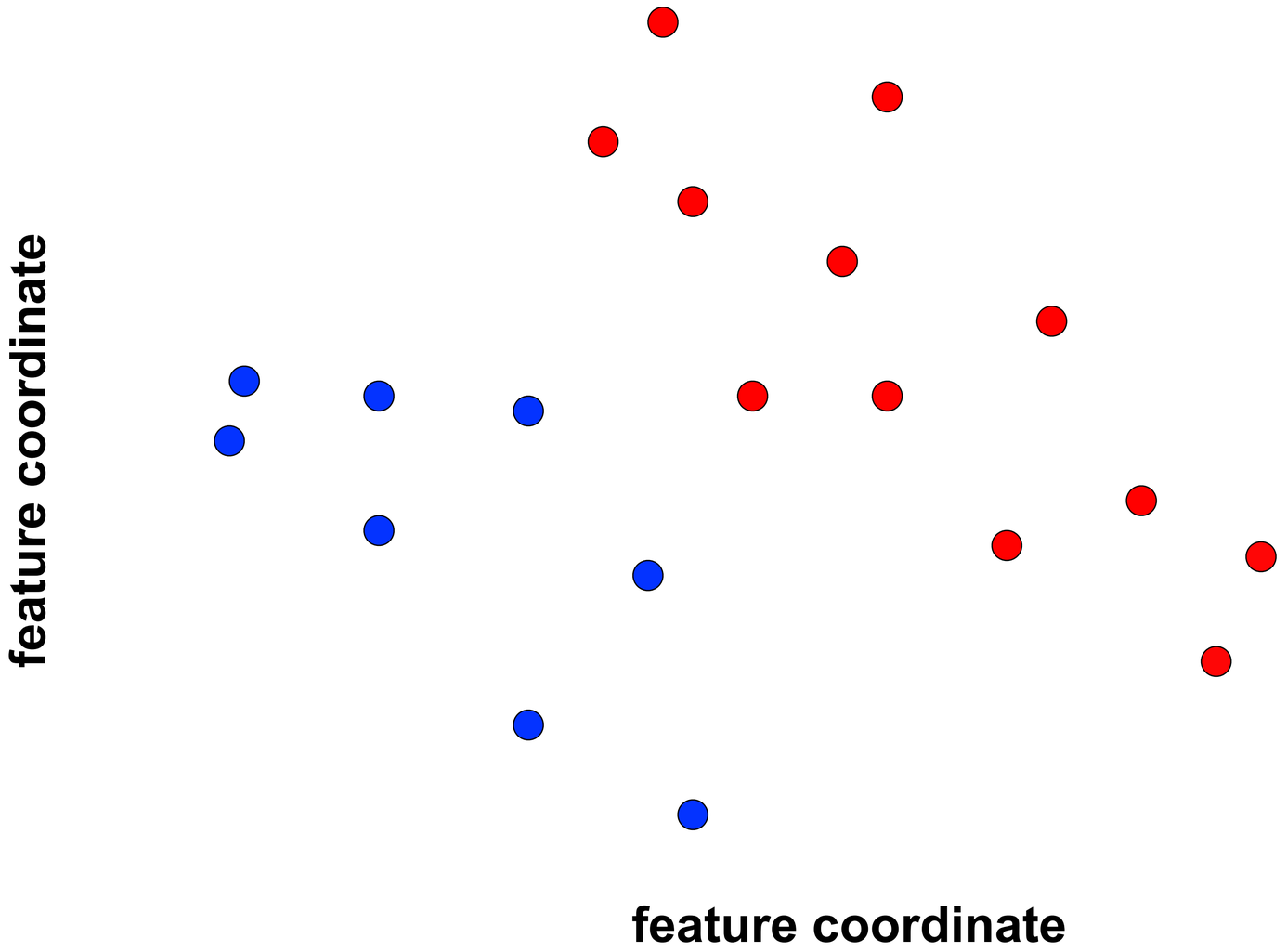
Assumption: Linear Function

$$y = f_{\mathbf{w}}(\mathbf{x}) = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

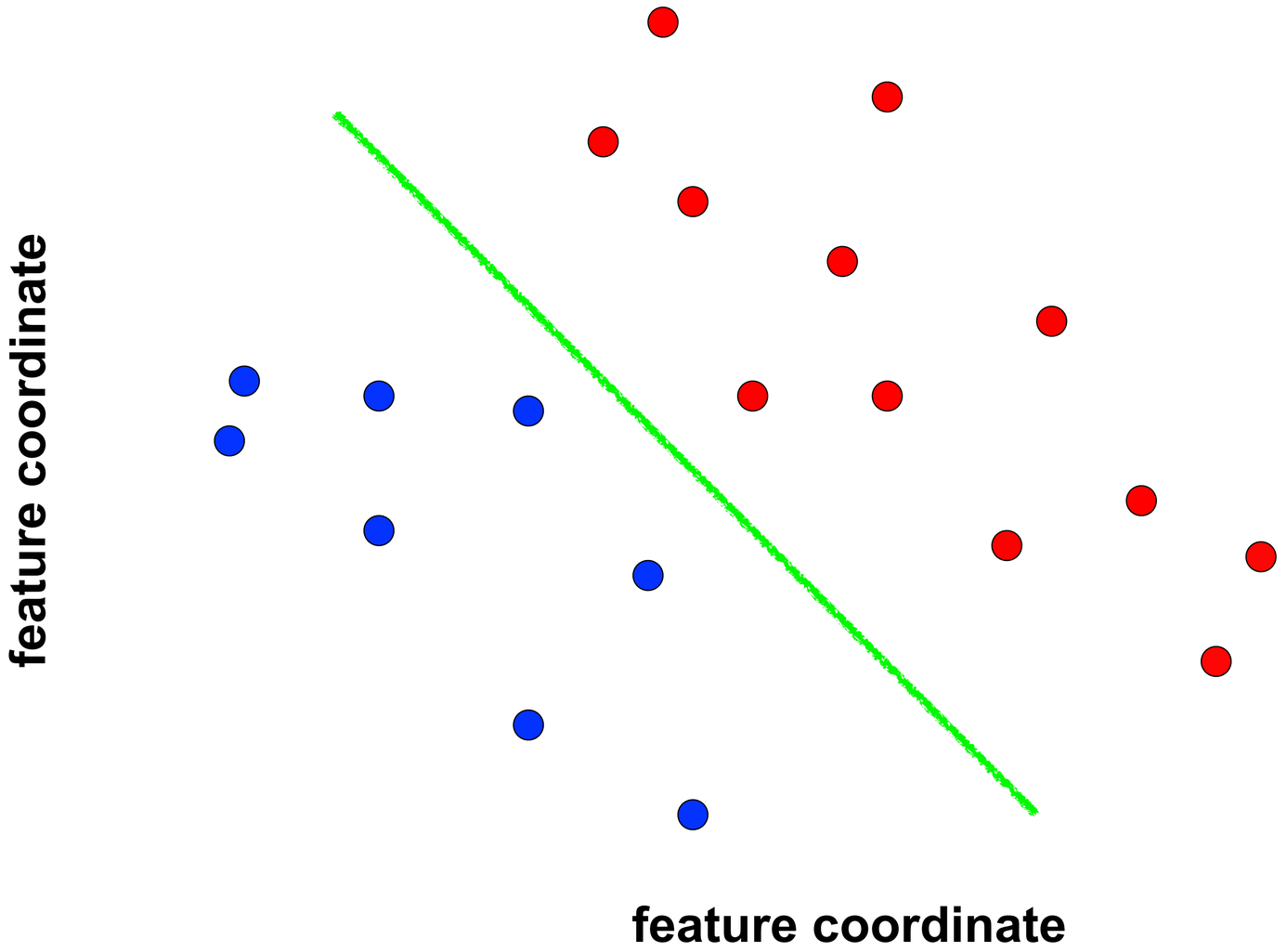
$$\mathbf{w}^T \mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{d=1}^{D} \mathbf{w}_d \mathbf{x}_d$$

$$\mathbf{x} \in \mathbb{R}^D, \mathbf{w} \in \mathbb{R}^D$$

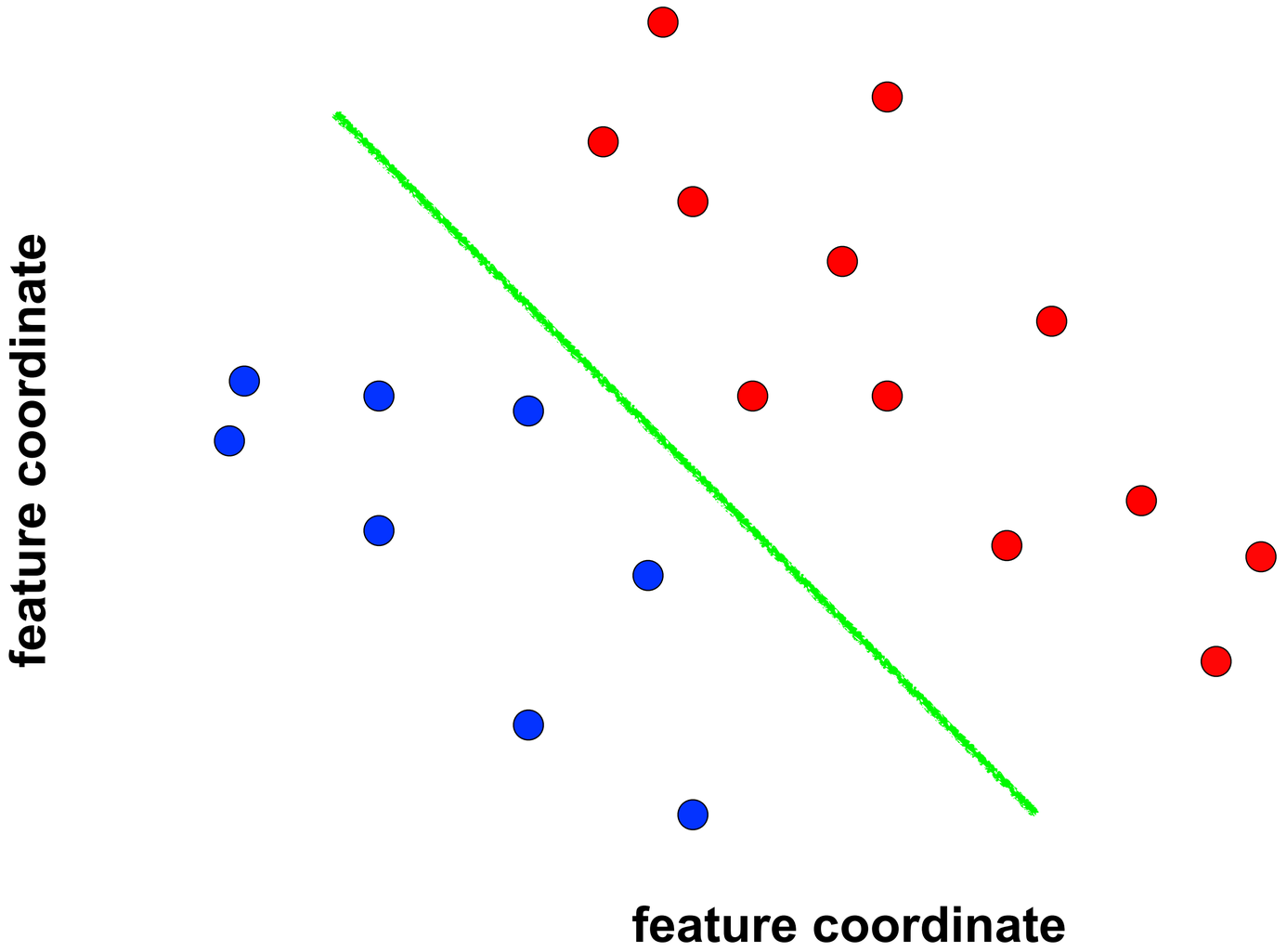
Reminder: Linear Classifier



Reminder: Linear Classifier



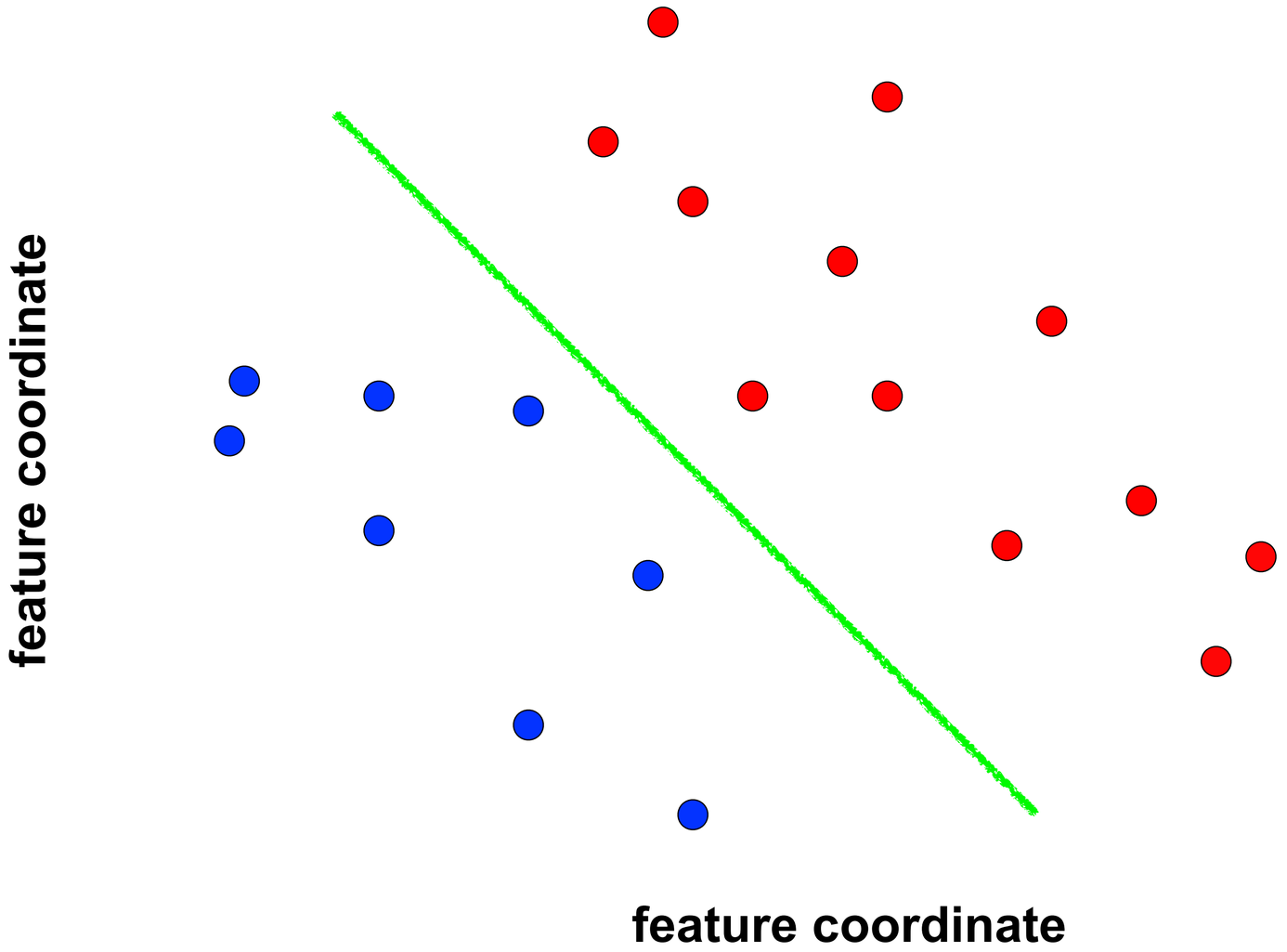
Reminder: Linear Classifier



$\mathbf{x}_i$ positive: $\mathbf{x}_i \cdot \mathbf{w} \geq 0$

$\mathbf{x}_i$ negative: $\mathbf{x}_i \cdot \mathbf{w} < 0$

Reminder: Linear Classifier



$\mathbf{x}_i$ positive: $\mathbf{x}_i \cdot \mathbf{w} \geq 0$

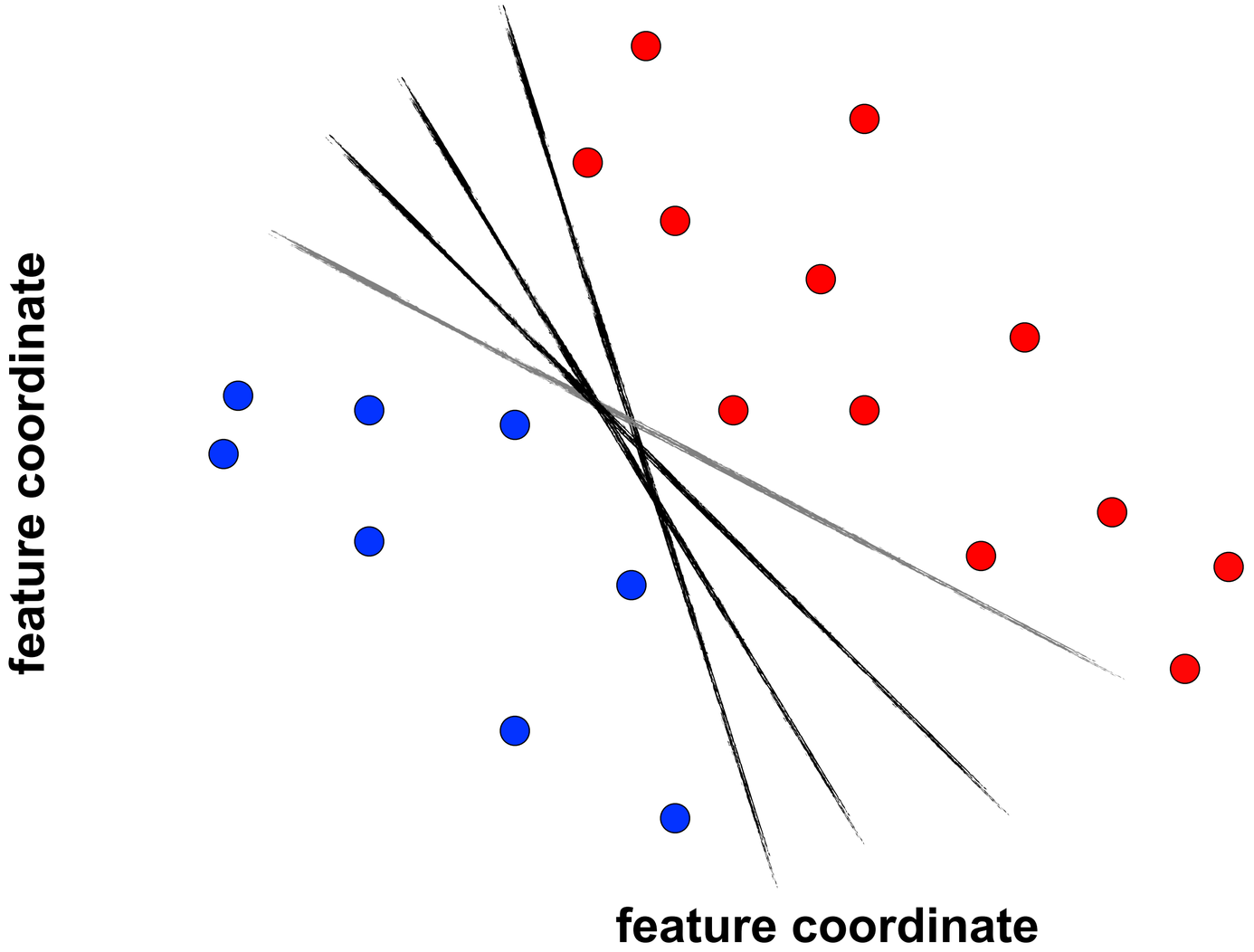
$\mathbf{x}_i$ negative: $\mathbf{x}_i \cdot \mathbf{w} < 0$

supervised setting

labelled input

$$y_t = \begin{cases} +1 & \bullet \\ -1 & \bullet \end{cases}$$

Which Line to Pick?



$\mathbf{x}_i$ positive: $\mathbf{x}_i \cdot \mathbf{w} \geq 0$

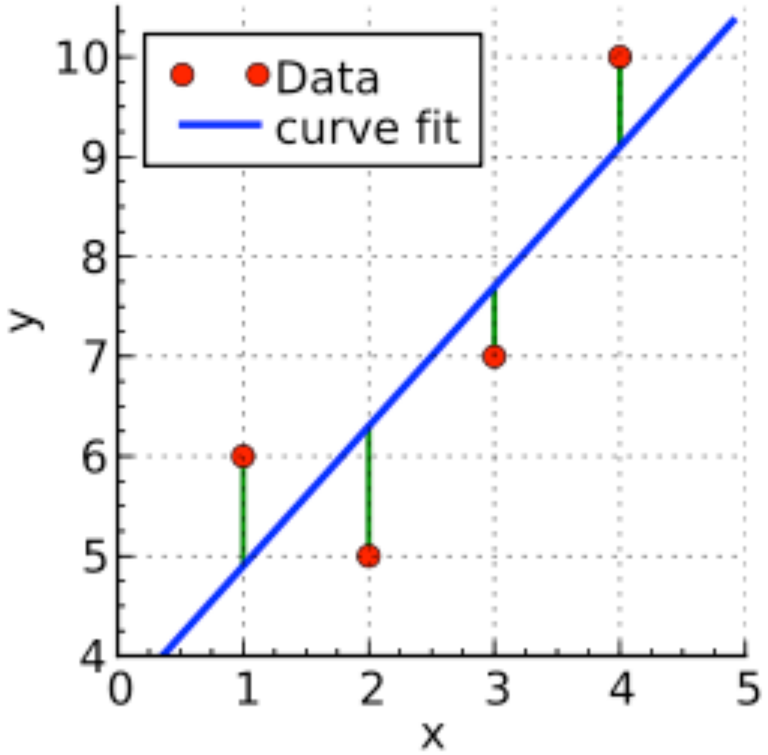
$\mathbf{x}_i$ negative: $\mathbf{x}_i \cdot \mathbf{w} < 0$

supervised setting

labelled input

$$y_t = \begin{cases} +1 & \bullet \\ -1 & \bullet \end{cases}$$

Linear Regression in 1D



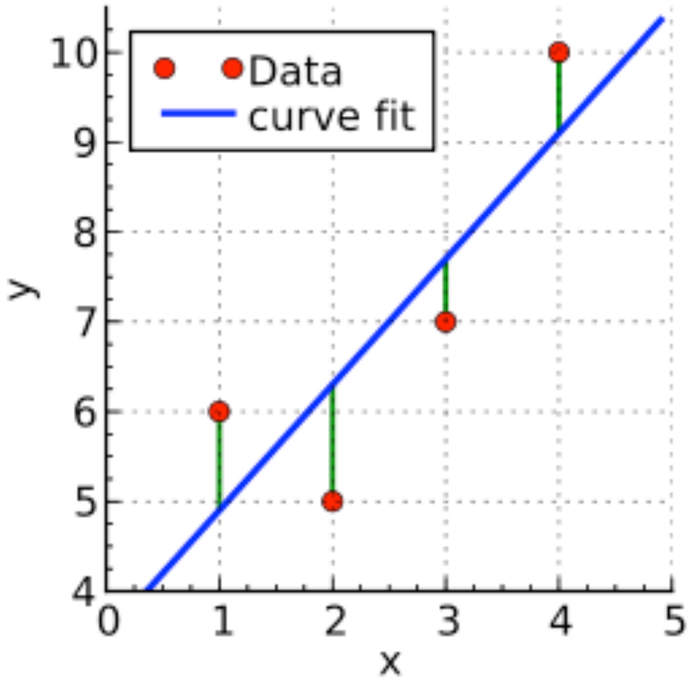
Training set: input–output pairs

$$\mathcal{S} = \{(x^i, y^i)\}, \quad i = 1 \ldots, N$$

$$x^i \in \mathbb{R}, \quad y^i \in \mathbb{R}$$

Linear regression in 1D

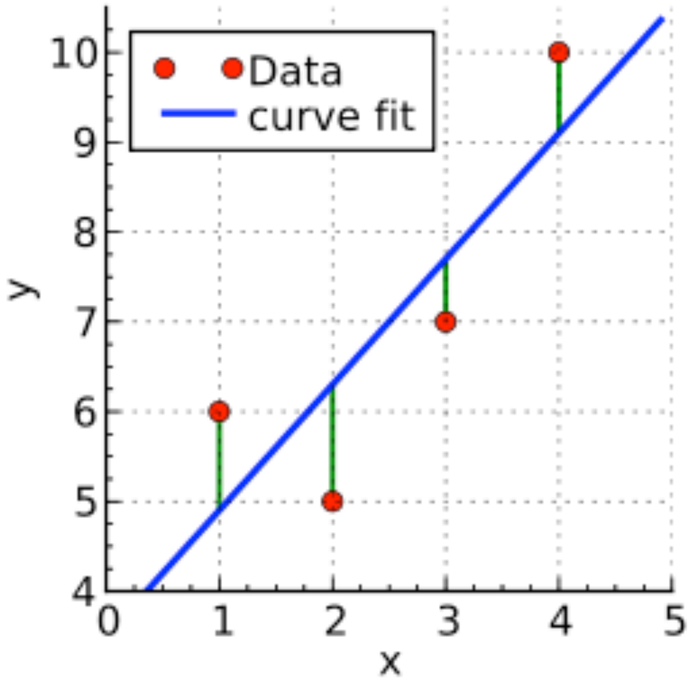
$$y^i = w_0 + w_1 x_1^i + \epsilon^i$$



Linear regression in 1D

$$y^i = w_0 + w_1 x_1^i + \epsilon^i$$

$$= w_0 x_0^i + w_1 x_1^i + \epsilon^i, \quad x_0^i = 1, \quad \forall i$$

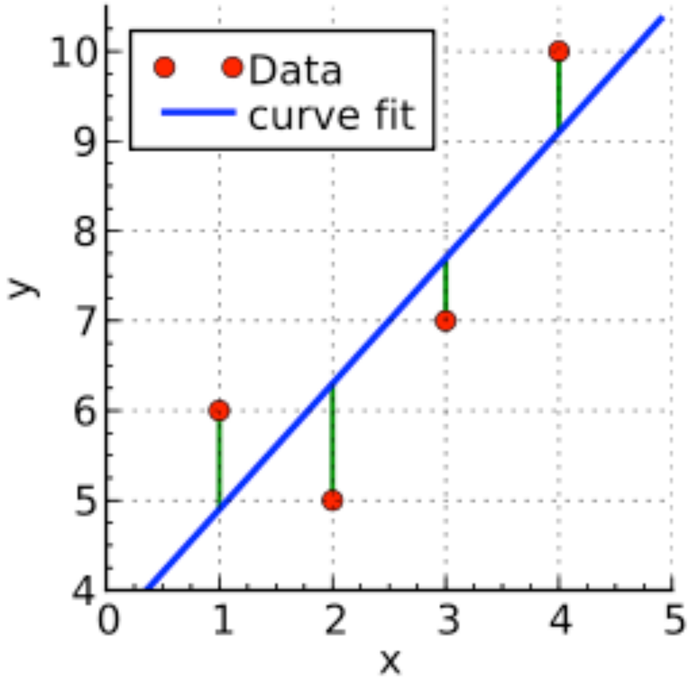


Linear regression in 1D

$$y^i = w_0 + w_1 x_1^i + \epsilon^i$$

w_0 bias

$$= w_0 x_0^i + w_1 x_1^i + \epsilon^i, \quad x_0^i = 1, \quad \forall i$$



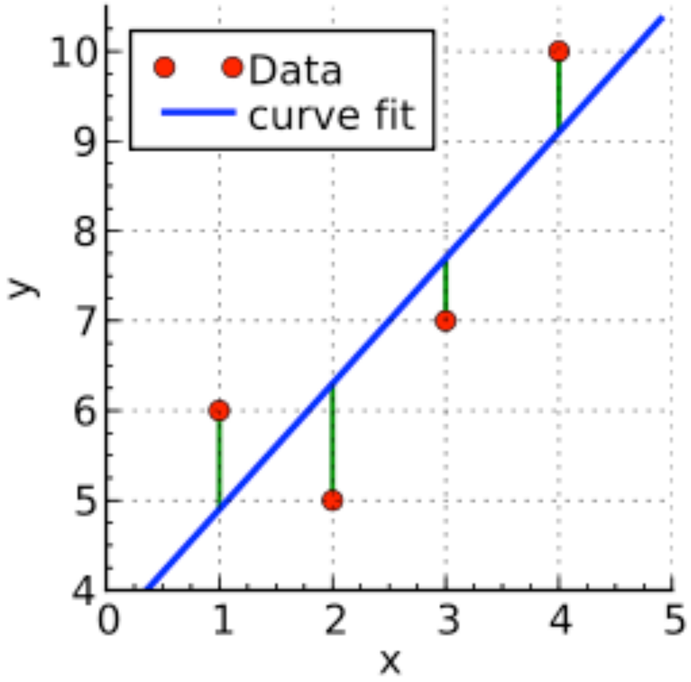
Linear regression in 1D

$$y^i = w_0 + w_1 x_1^i + \epsilon^i$$

w_0 bias

$$= w_0 x_0^i + w_1 x_1^i + \epsilon^i, \quad x_0^i = 1, \quad \forall i$$

$$= \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$



Linear regression in 1D

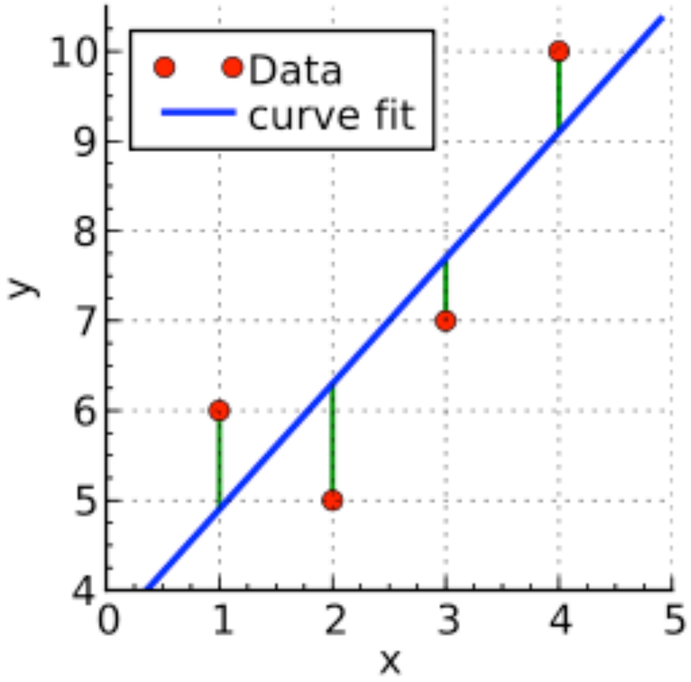
$$y^i = w_0 + w_1 x_1^i + \epsilon^i$$

w_0 bias

$$= w_0 x_0^i + w_1 x_1^i + \epsilon^i, \quad x_0^i = 1, \quad \forall i$$

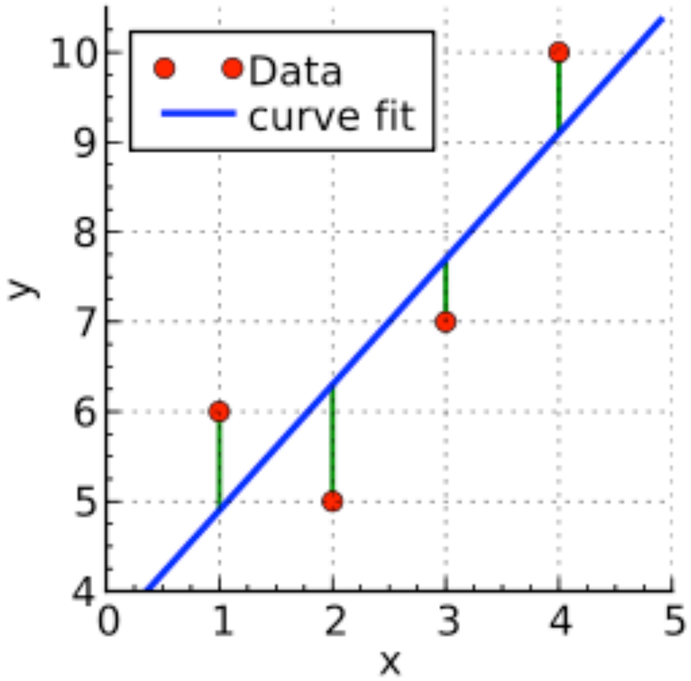
$$= \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

noise



Sum of Square Errors *(MSE without the mean)*

$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

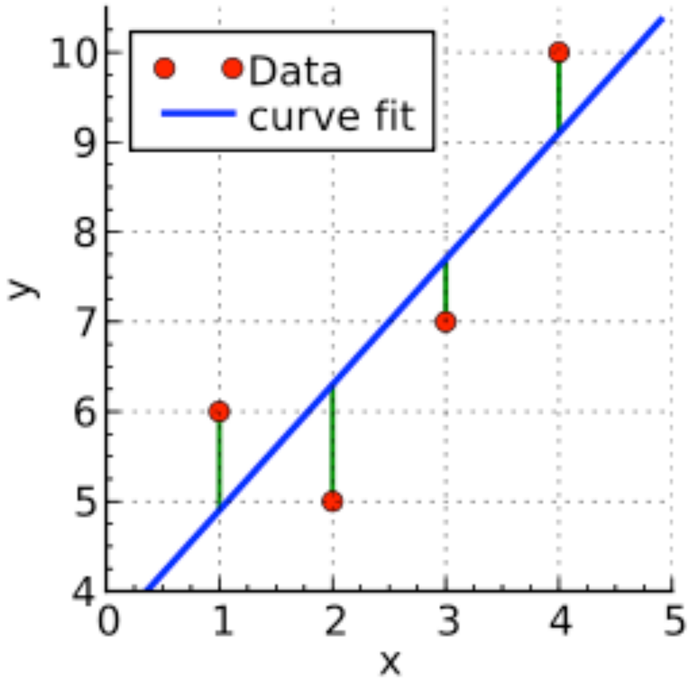


Sum of Square Errors *(MSE without the mean)*

$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

$$L(\mathbf{w}) = \sum_{i=1}^{N} (\epsilon^i)^2$$



Sum of Square Errors *(MSE without the mean)*

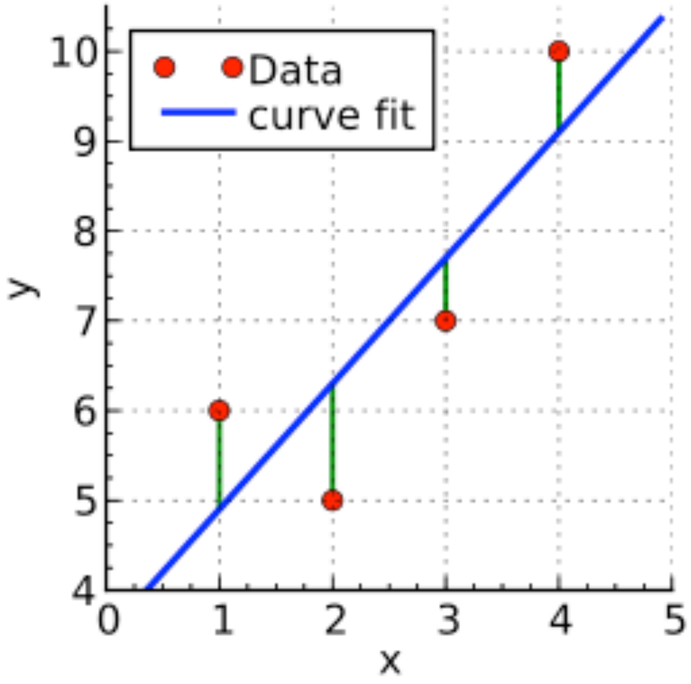
$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

$$L(\mathbf{w}) = \sum_{i=1}^{N} (\epsilon^i)^2$$

In two variables:

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$



Sum of Square Errors *(MSE without the mean)*

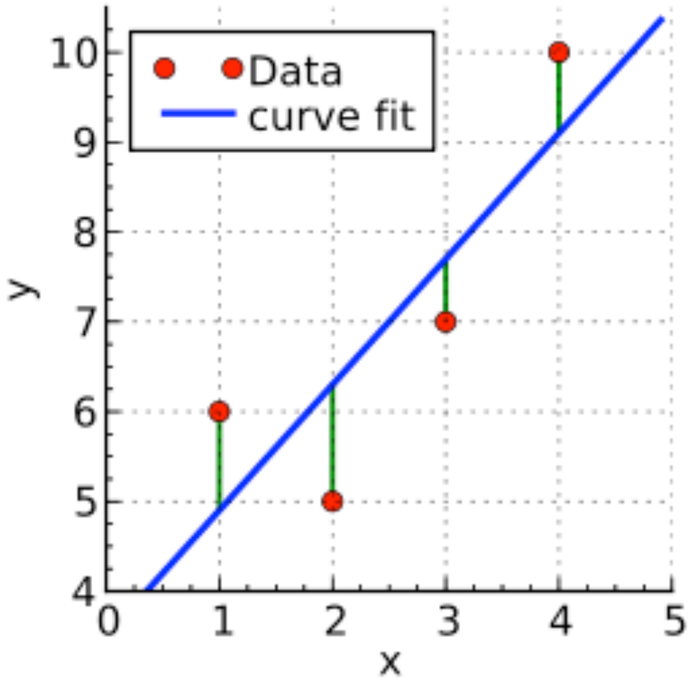
$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

$$L(\mathbf{w}) = \sum_{i=1}^{N} (\epsilon^i)^2$$

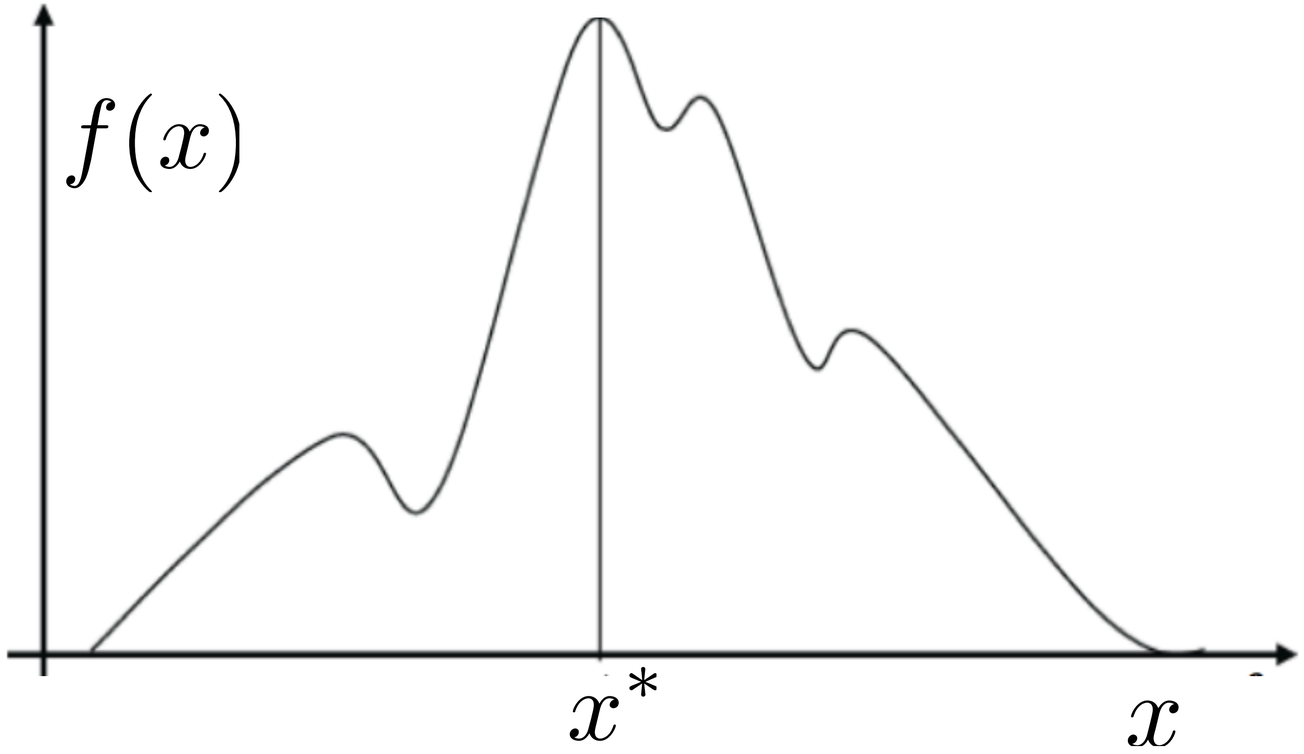
In two variables:

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

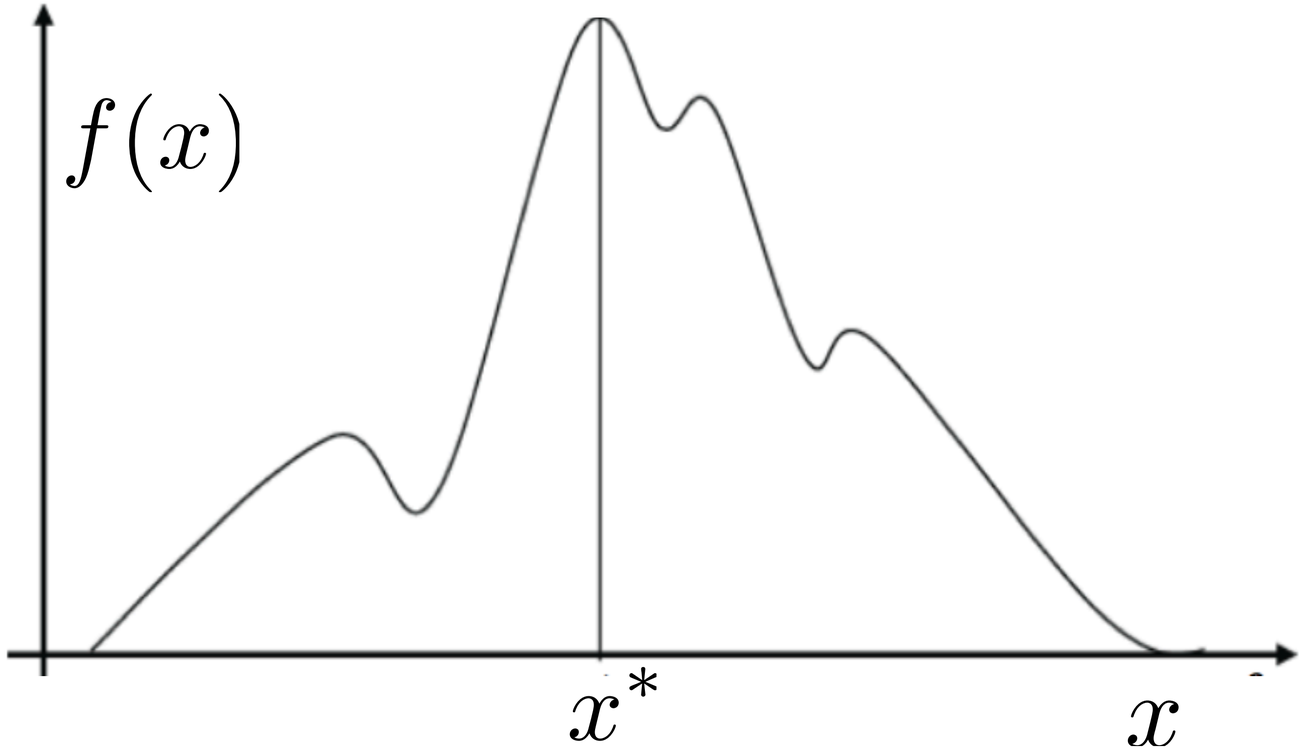


Question: what is the best (or least bad) value of ***w****?*

Calculus 101

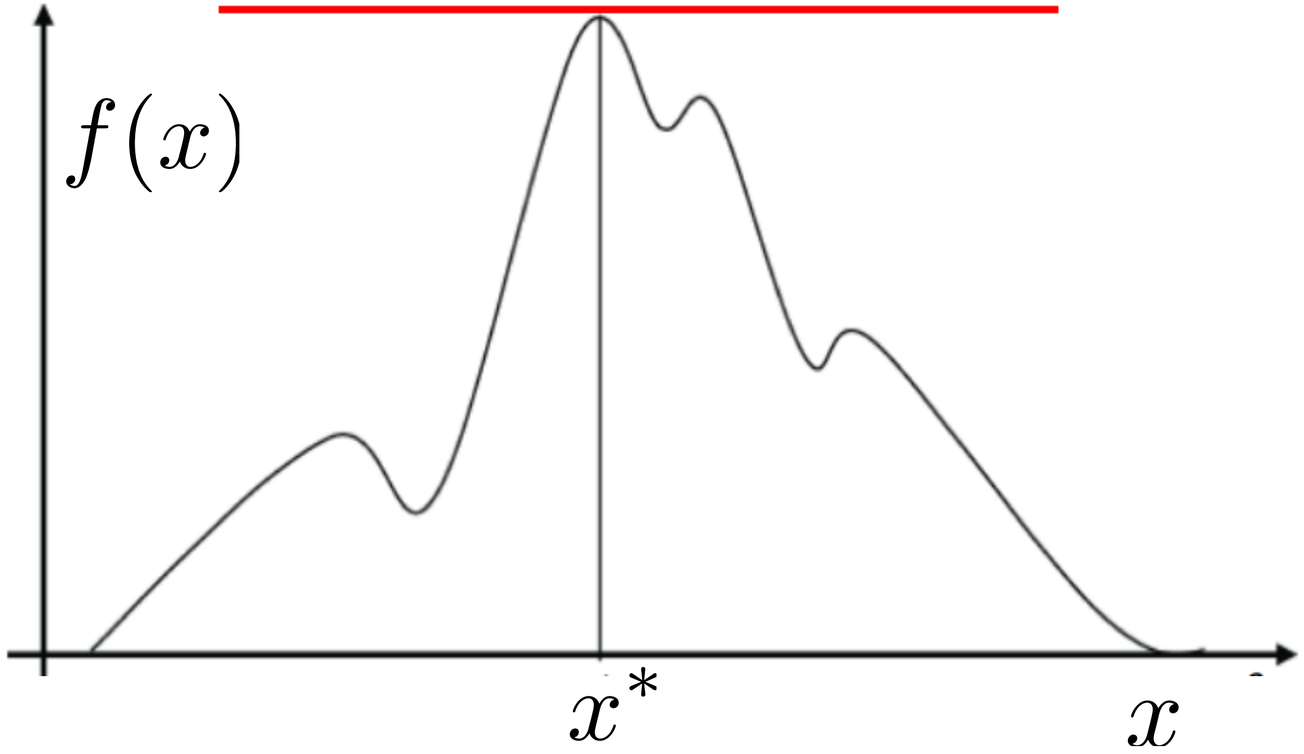


Calculus 101



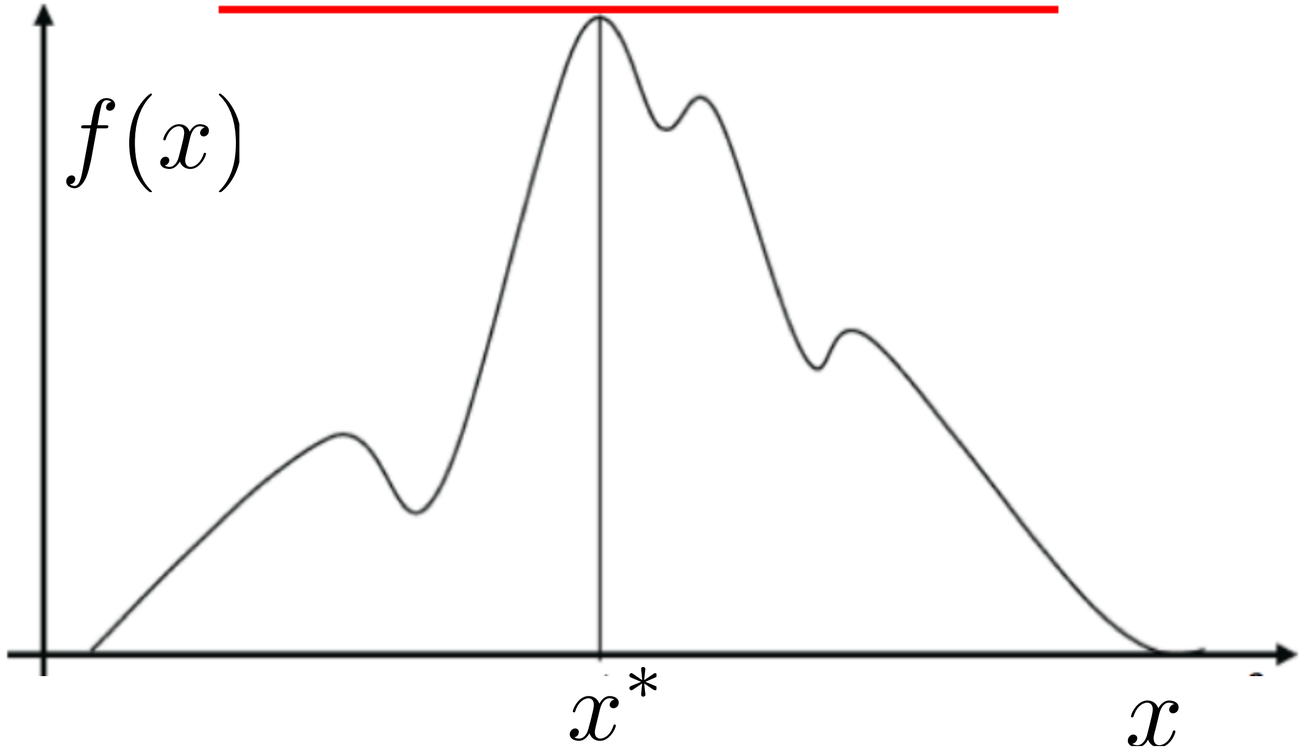
$$x^* = \text{argmax}_x f(x)$$

Local Extrema Condition



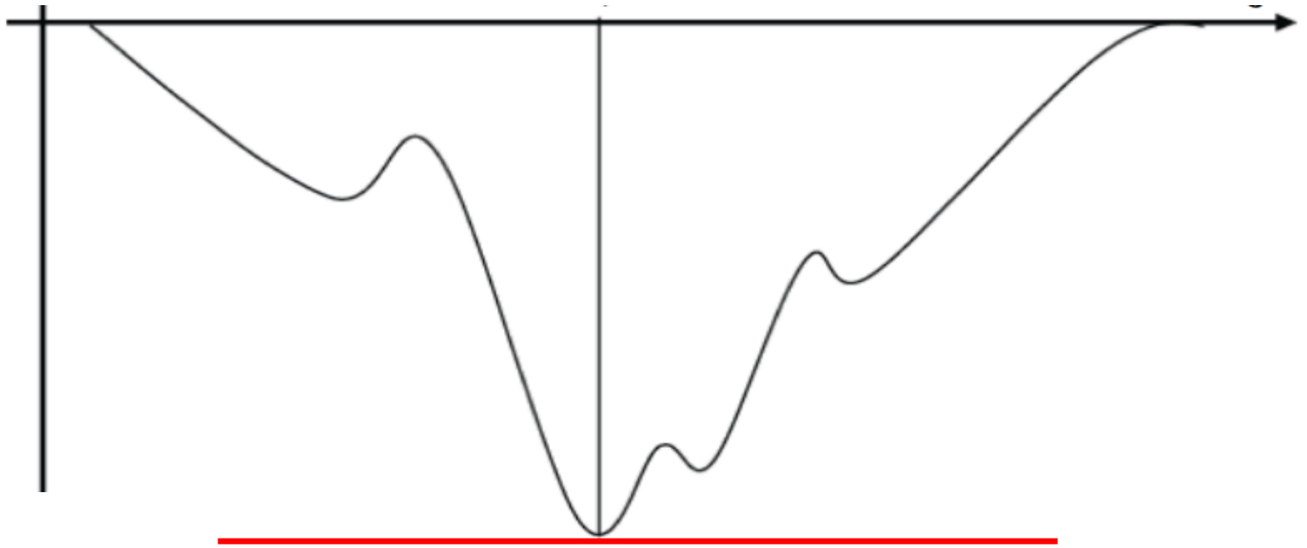
$$x^* = \text{argmax}_x f(x)$$

Local Extrema Condition



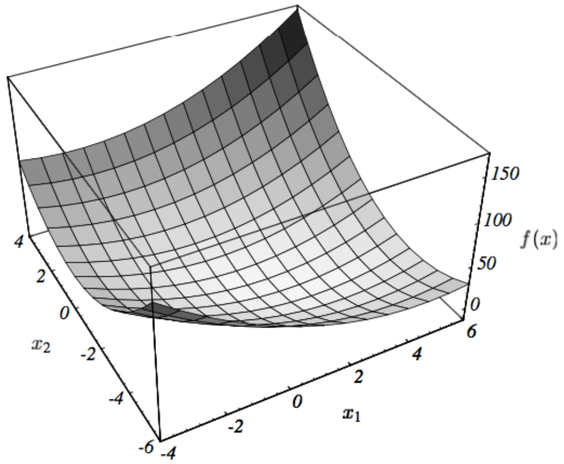
$$x^* = \text{argmax}_x f(x) \quad \rightarrow \quad f'(x^*) = 0$$

Local Extrema Condition



$$x^* = \text{argmax}_x f(x) \quad \rightarrow \quad f'(x^*) = 0$$

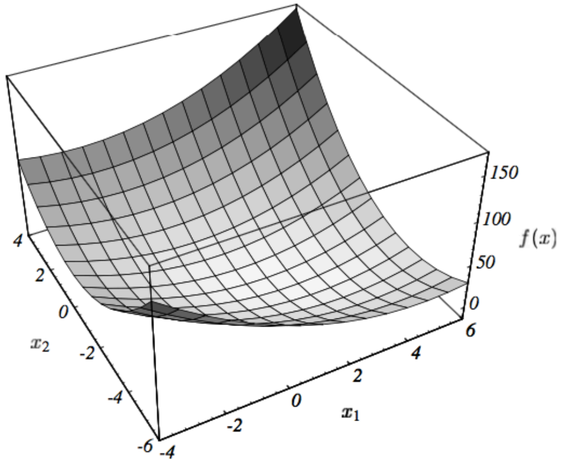
Vector Calculus 101

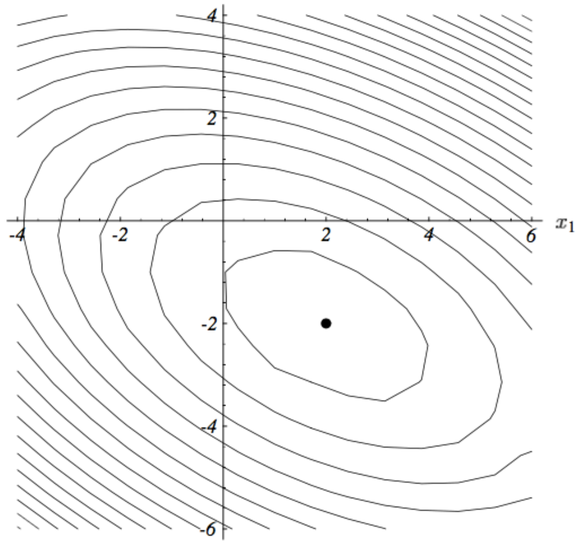


$$f(\mathbf{x})$$

2D function graph

Vector Calculus 101





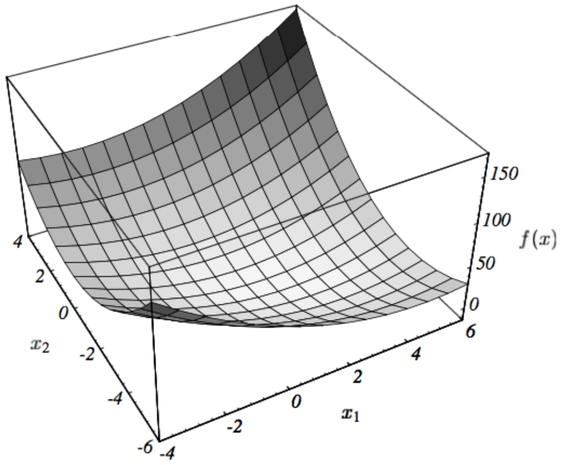
$$f(\mathbf{x})$$

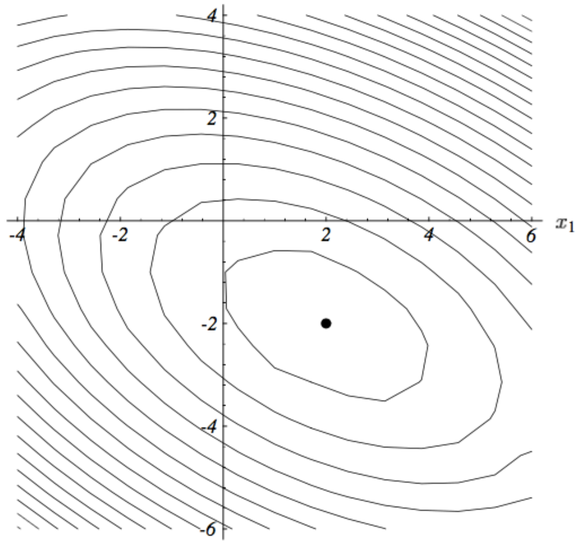
$$f(\mathbf{x}) = c$$

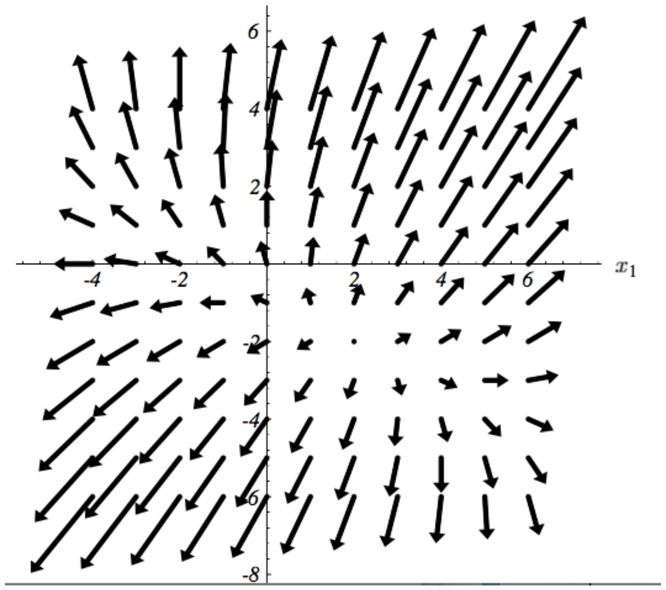
2D function graph

isocontours

Vector Calculus 101







$f(\mathbf{x})$

$f(\mathbf{x}) = c$

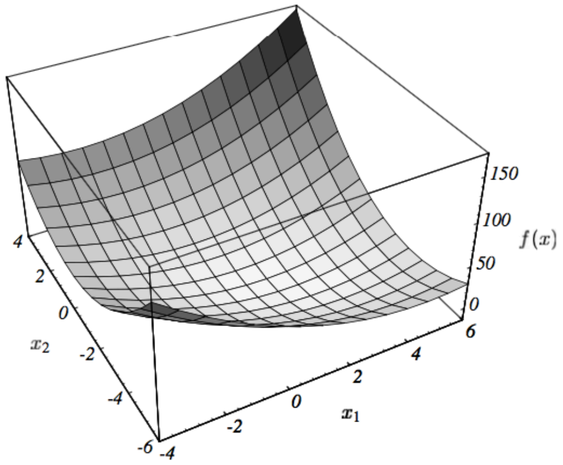
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

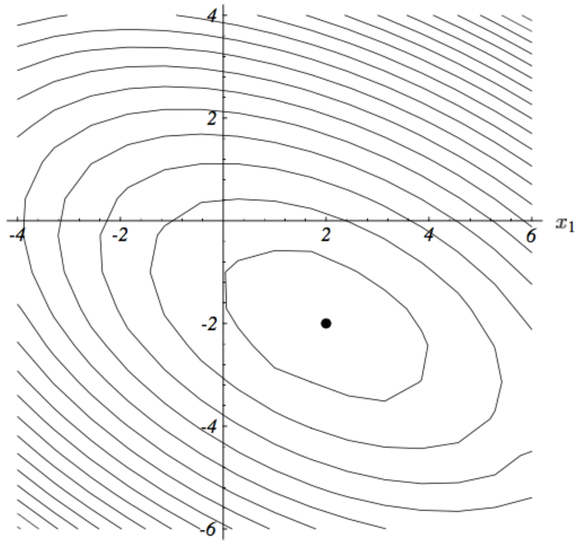
2D function graph

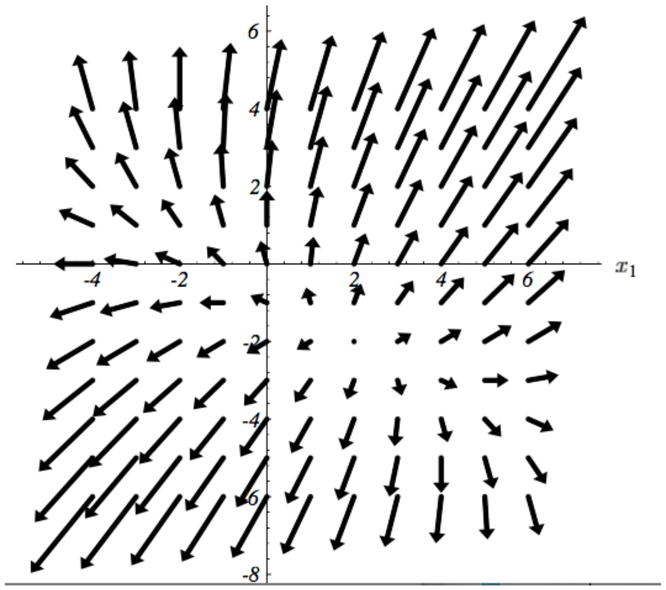
isocontours

gradient field

Vector Calculus 101







$$f(\mathbf{x})$$

$$f(\mathbf{x}) = c$$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

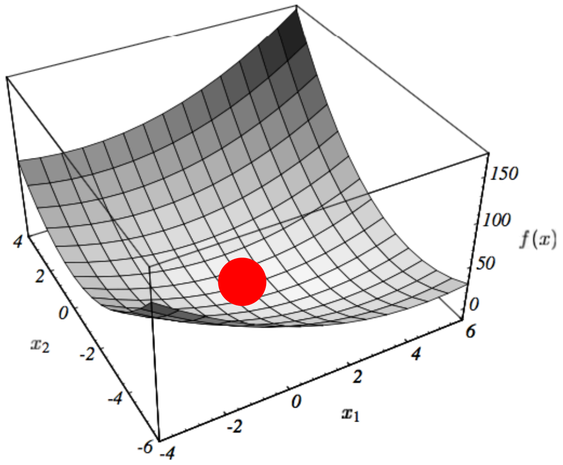
2D function graph

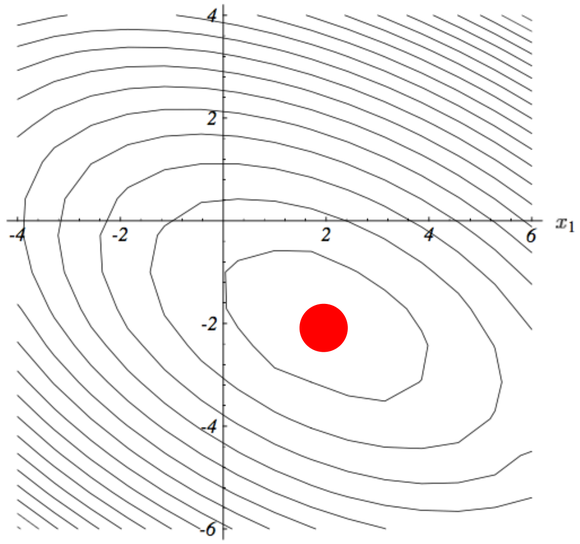
isocontours

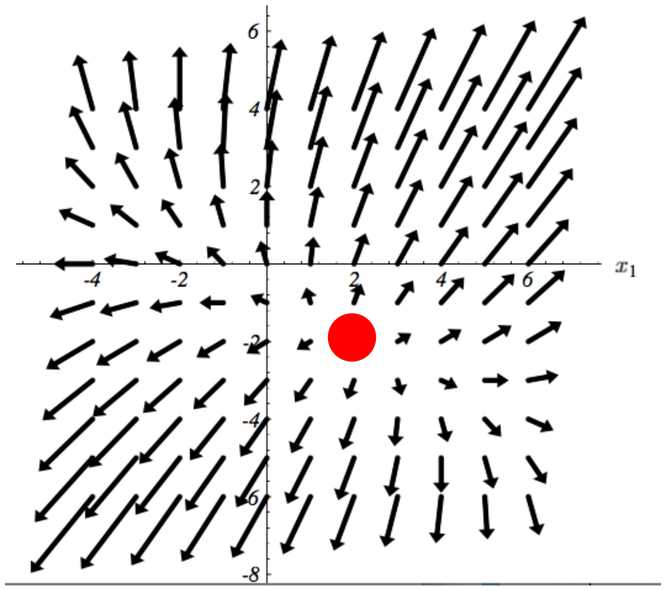
gradient field

- at minimum of function: $\nabla f(\mathbf{x}) = \mathbf{0}$

Vector Calculus 101







$f(\mathbf{x})$

$f(\mathbf{x}) = c$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

2D function graph

isocontours

gradient field

- at minimum of function: $\nabla f(\mathbf{x}) = \mathbf{0}$

Line Fitting

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

Line Fitting

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

Line Fitting

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0}$$

Line Fitting

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [z^i]^2}{\partial z^i} \frac{\partial z^i}{\partial w_0}$$

Line Fitting

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [z^i]^2}{\partial z^i} \frac{\partial z^i}{\partial w_0}$$

$$z^i = y^i - \left(w_0 x_0^i + w_1 x_1^i \right)$$

Line Fitting

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [z^i]^2}{\partial z^i} \frac{\partial z^i}{\partial w_0} = \sum_{i=1}^{N} (2z^i)(-x_0^i)$$

$$z^i = y^i - (w_0 x_0^i + w_1 x_1^i)$$

Line Fitting

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [z^i]^2}{\partial z^i} \frac{\partial z^i}{\partial w_0} = \sum_{i=1}^{N} (2z^i)(-x_0^i)$$

$$= -2 \sum_{i=1}^{N} (y^i - (w_0 x_0^i + w_1 x_1^i)) x_0^i$$

$$z^i = y^i - (w_0 x_0^i + w_1 x_1^i)$$

Line Fitting (continued)

Line Fitting (continued)

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

Line Fitting (continued)

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

$$= -2 \sum_{i=1}^{N} \left(y^i x_0^i - w_0 x_0^i x_0^i - w_1 x_1^i x_0^i \right)$$

Line Fitting (continued)

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

$$= -2 \sum_{i=1}^{N} \left(y^i x_0^i - w_0 x_0^i x_0^i - w_1 x_1^i x_0^i \right)$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = 0$$

Line Fitting (continued)

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2}{\partial w_0}$$

$$= -2 \sum_{i=1}^{N} \left(y^i x_0^i - w_0 x_0^i x_0^i - w_1 x_1^i x_0^i \right)$$

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = 0$$

$$\sum_{i=1}^{N} y^i x_0^i = w_0 \sum_{i=1}^{N} x_0^i x_0^i + w_1 \sum_{i=1}^{N} x_1^i x_0^i$$

Line Fitting (continued)

$$\sum_{i=1}^{N} y^i x_0^i = w_0 \sum_{i=1}^{N} x_0^i x_0^i + w_1 \sum_{i=1}^{N} x_1^i x_0^i$$

$$\sum_{i=1}^{N} y^i x_1^i = w_0 \sum_{i=1}^{N} x_0^i x_1^i + w_1 \sum_{i=1}^{N} x_1^i x_1^i$$

Line Fitting (continued)

$$\sum_{i=1}^{N} y^i x_0^i = w_0 \sum_{i=1}^{N} x_0^i x_0^i + w_1 \sum_{i=1}^{N} x_1^i x_0^i$$

$$\sum_{i=1}^{N} y^i x_1^i = w_0 \sum_{i=1}^{N} x_0^i x_1^i + w_1 \sum_{i=1}^{N} x_1^i x_1^i$$

2x2 system of equations

$$\begin{bmatrix} \sum_{i=1}^{N} y^i x_0^i \\ \sum_{i=1}^{N} y^i x_1^i \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\ \sum_{i=1}^{N} x_0^i x_1^i & \sum_{i=1}^{N} x_1^i x_1^i \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

Line Fitting (continued)

$$\begin{bmatrix} \sum_{i=1}^{N} y^i x_0^i \\ \sum_{i=1}^{N} y^i x_1^i \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\ \sum_{i=1}^{N} x_0^i x_1^i & \sum_{i=1}^{N} x_1^i x_1^i \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

Line Fitting (continued)

$$\begin{bmatrix} \sum_{i=1}^{N} y^i x_0^i \\ \sum_{i=1}^{N} y^i x_1^i \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\ \sum_{i=1}^{N} x_0^i x_1^i & \sum_{i=1}^{N} x_1^i x_1^i \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w}$$

$$\mathbf{y} = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_0^1 & x_1^1 \\ \vdots & \vdots \\ x_0^N & x_2^N \end{bmatrix}$$

Line Fitting (continued)

$$\begin{bmatrix} \sum_{i=1}^{N} y^i x_0^i \\ \sum_{i=1}^{N} y^i x_1^i \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\ \sum_{i=1}^{N} x_0^i x_1^i & \sum_{i=1}^{N} x_1^i x_1^i \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w}$$

$$\mathbf{y} = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_0^1 & x_1^1 \\ \vdots & \vdots \\ x_0^N & x_2^N \end{bmatrix}$$

$$\boxed{\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

Line Fitting (continued)

$$\begin{bmatrix} \sum_{i=1}^{N} y^i x_0^i \\ \sum_{i=1}^{N} y^i x_1^i \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\ \sum_{i=1}^{N} x_0^i x_1^i & \sum_{i=1}^{N} x_1^i x_1^i \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

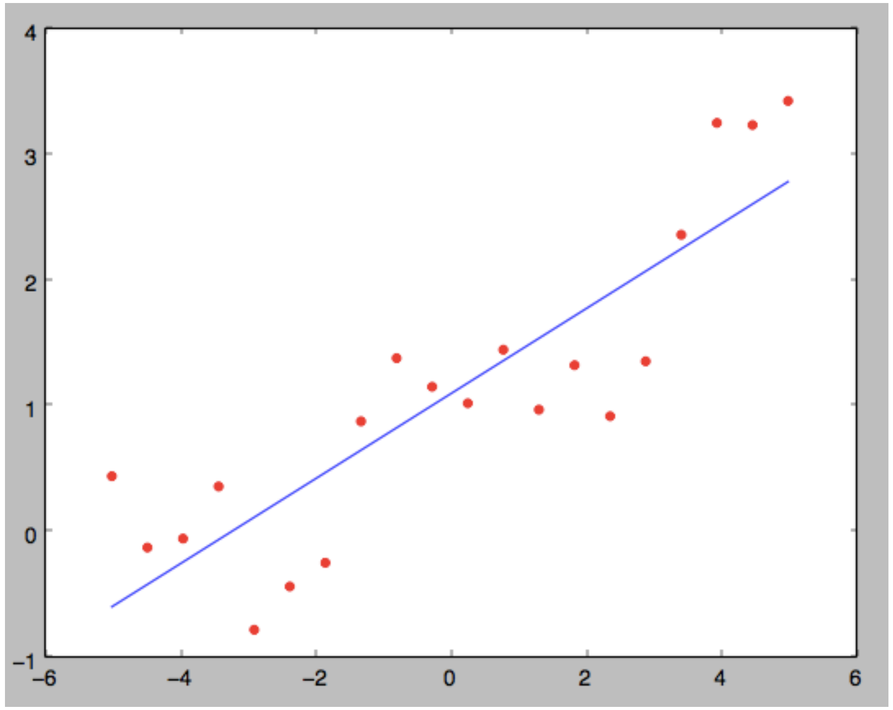
$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w}$$

$$\mathbf{y} = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_0^1 & x_1^1 \\ \vdots & \vdots \\ x_0^N & x_2^N \end{bmatrix}$$

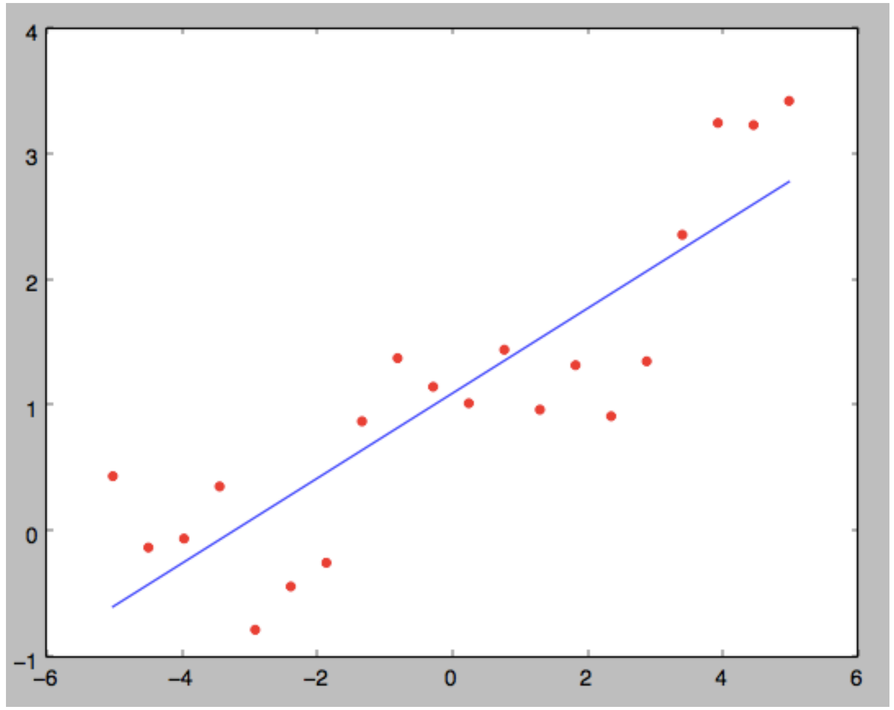
Normal Equation

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

Code Example

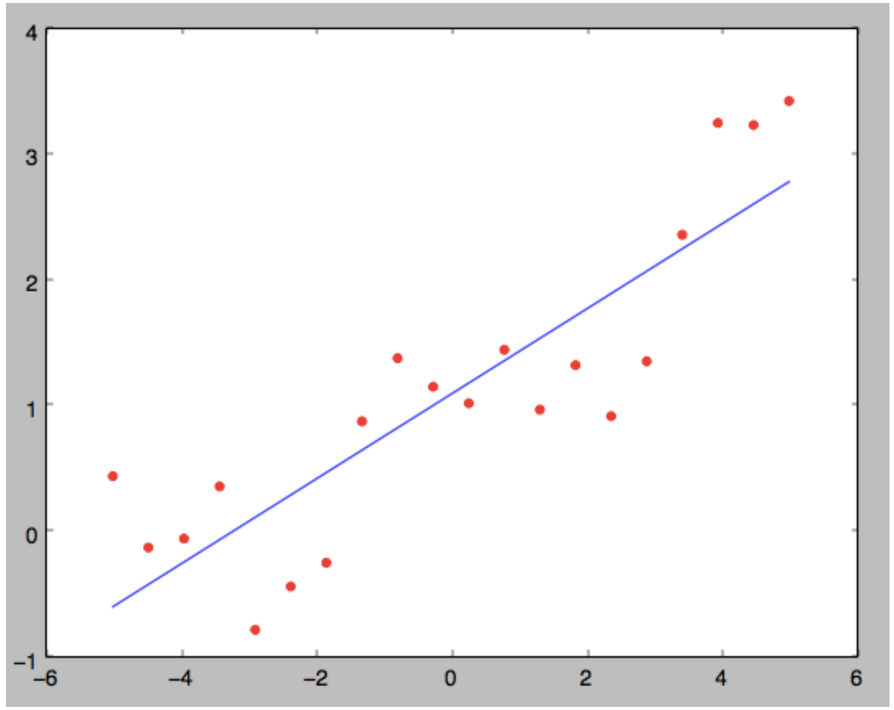


Code Example



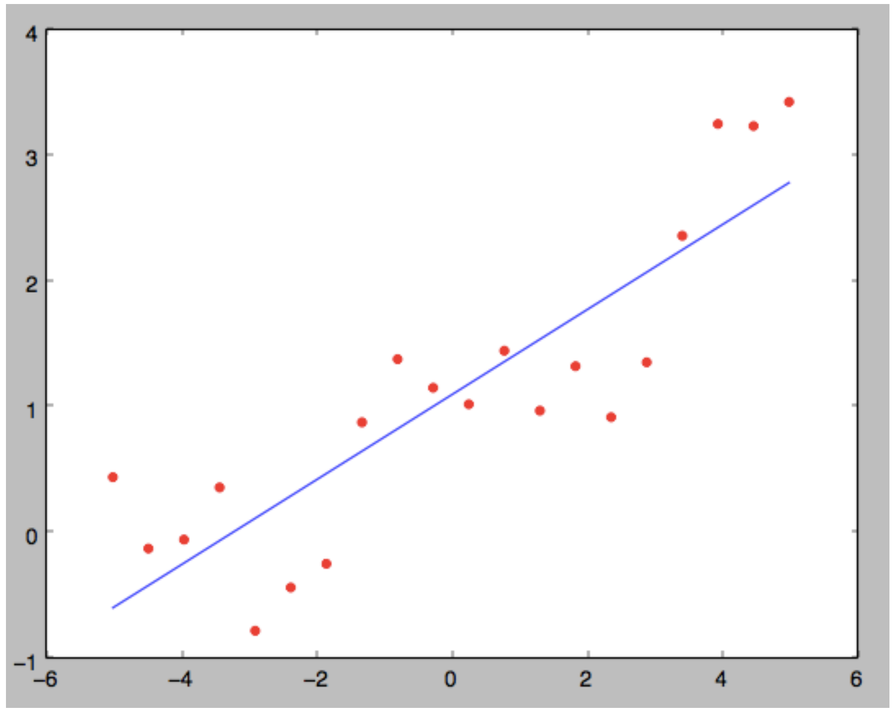

```
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 2
w = rand(2,1) # w[0] is random constant term (offset from origin), w[1] is random linear term (slope)
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# For ridge regression, use regularizer
#weight = 0.01
#w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x

# visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()
```

Code Example




```python
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 2
w = rand(2,1) # w[0] is random constant term (offset from origin), w[1] is random linear term (slope)
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# For ridge regression, use regularizer
#weight = 0.01
#w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x

# visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()
```

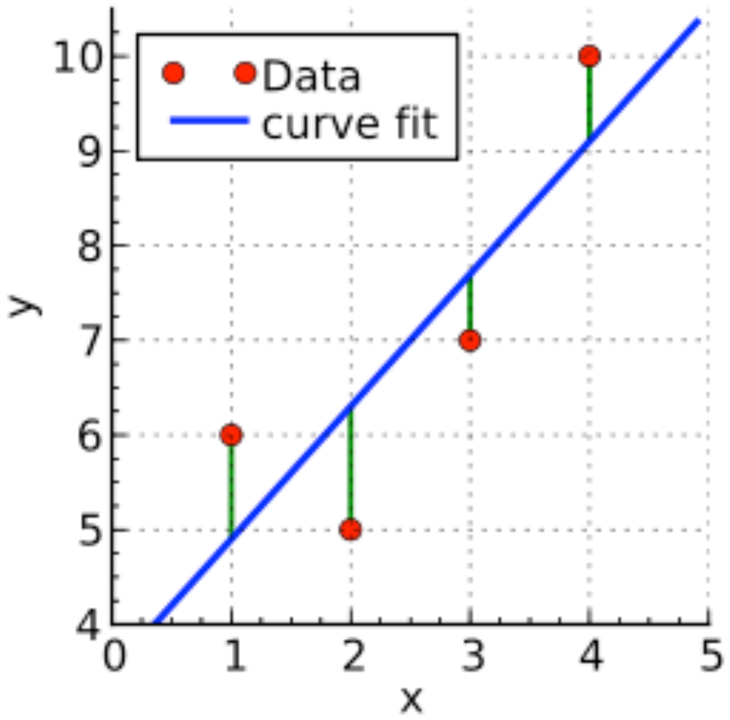
Code Example



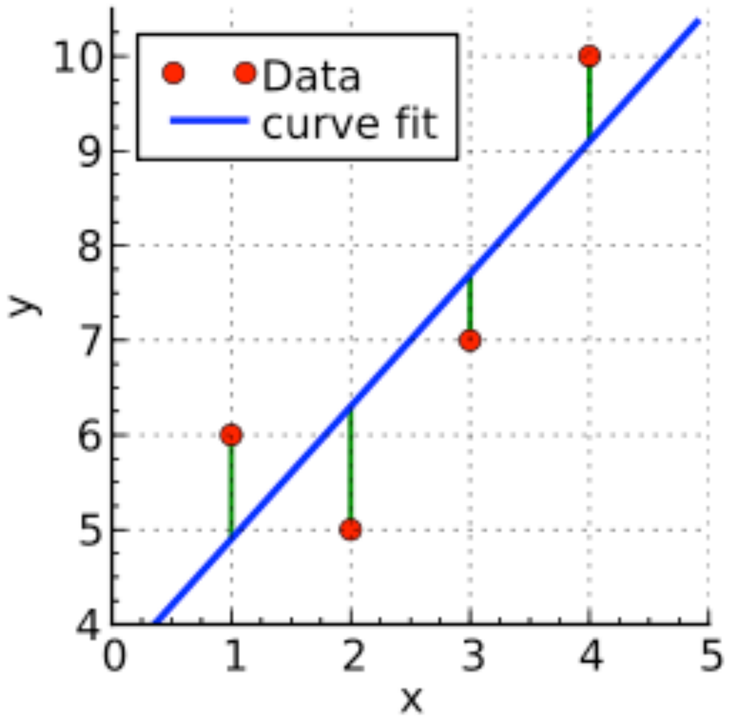

```
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 2
w = rand(2,1) # w[0] is random constant term (offset from origin), w[1] is random linear term (slope)
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# For ridge regression, use regularizer
#weight = 0.01
#w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x

# visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()
```

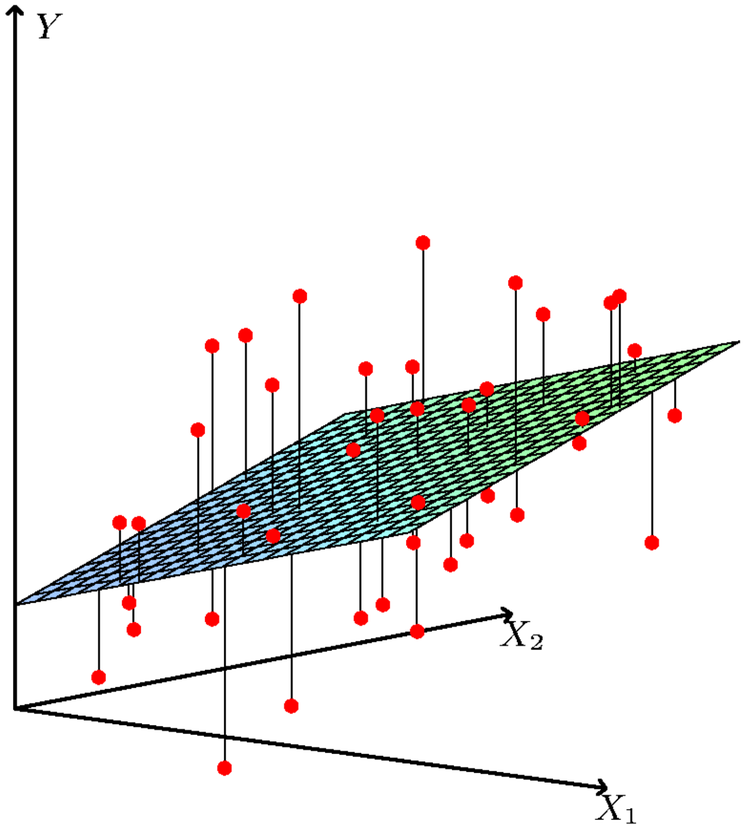
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Linear Regression (Line/Plane Fitting)



Linear Regression (Line/Plane Fitting)





LS Solution for Regression

$$L(\mathbf{w}) = \sum_{i=1}^{N} (y^i - \mathbf{w}^T \mathbf{x}^i)^2 = \sum_{i=1}^{N} (\epsilon^i)^2$$

$$L(\mathbf{w}) = \begin{bmatrix} \epsilon^1 & \epsilon^2 & \dots & \epsilon^N \end{bmatrix} \begin{bmatrix} \epsilon^1 \\ \epsilon^2 \\ \vdots \\ \epsilon^N \end{bmatrix}$$

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$$

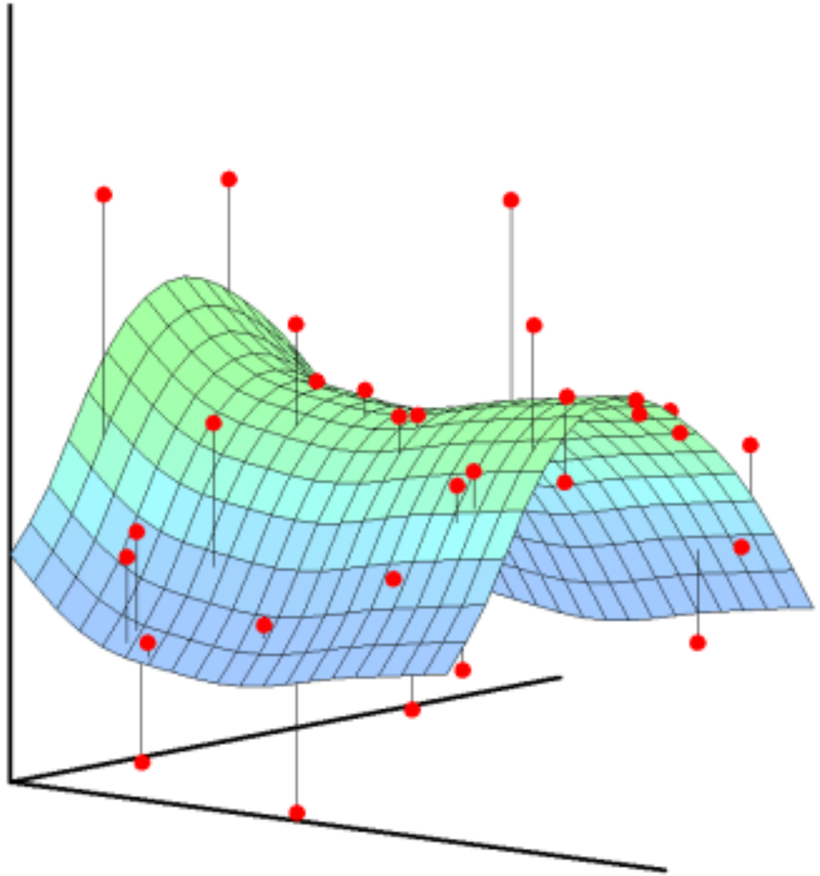
LS Solution for Regression

$$L(\mathbf{w}) = \sum_{i=1}^{N} (y^i - \mathbf{w}^T \mathbf{x}^i)^2 = \sum_{i=1}^{N} (\epsilon^i)^2$$

$$L(\mathbf{w}) = \begin{bmatrix} \epsilon^1 & \epsilon^2 & \dots & \epsilon^N \end{bmatrix} \begin{bmatrix} \epsilon^1 \\ \epsilon^2 \\ \vdots \\ \epsilon^N \end{bmatrix}$$

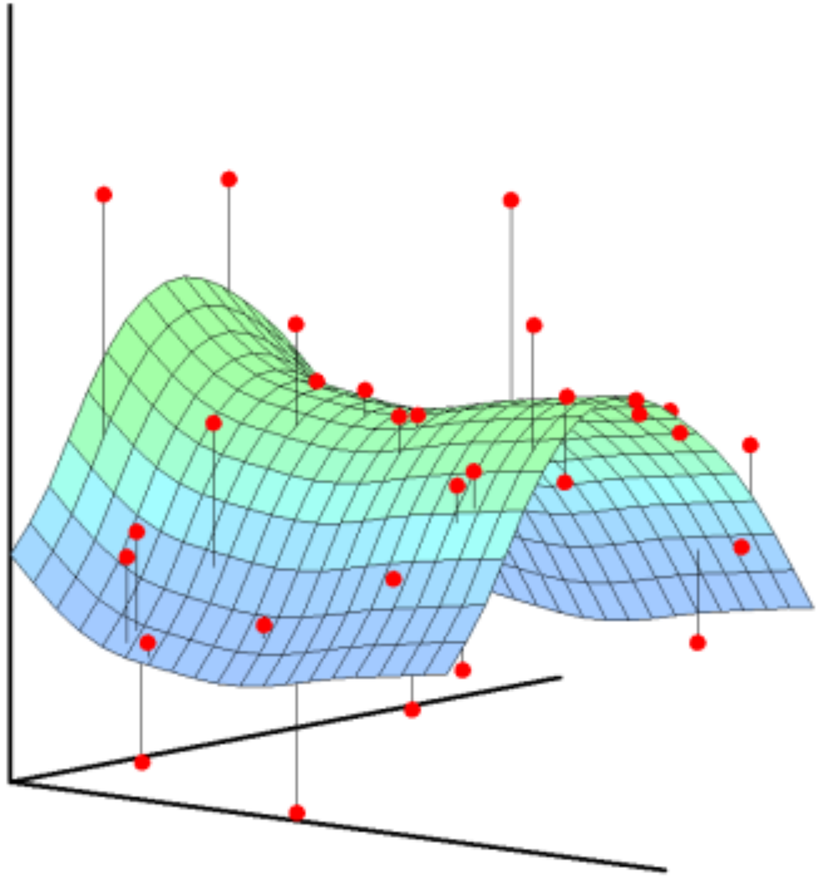
$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \qquad \mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$$

Generalized Linear Regression



$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}$$

Generalized Linear Regression

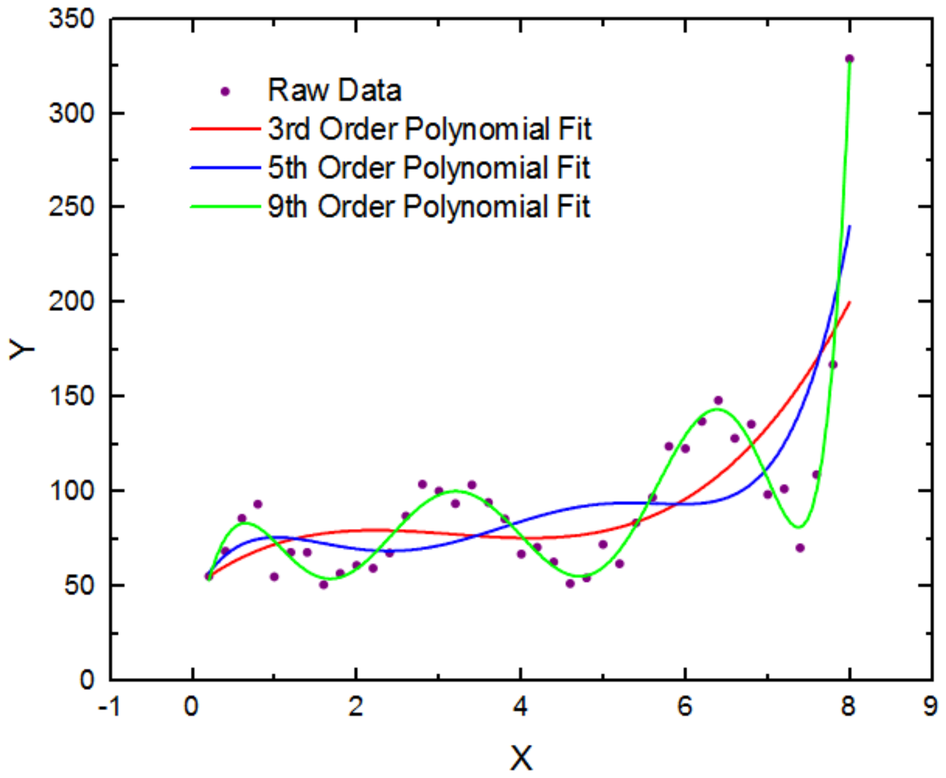


$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}$$

known nonlinearity

1D Example: k-th Degree Polynomial Fitting

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} 1 \\ x \\ \vdots \\ (x)^K \end{bmatrix}$$



$$\langle \mathbf{w}, \boldsymbol{\phi}(x) \rangle = w_0 + w_1 x + \ldots + w_k (x)^K$$

Generalized Linear Regression

$$L(\mathbf{w}) = \sum_{i=1}^{N} (y^i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}^i))^T = \sum_{i=1}^{N} (\epsilon^i)^2$$

Generalized Linear Regression

$$L(\mathbf{w}) = \sum_{i=1}^{N} (y^i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}^i))^T = \sum_{i=1}^{N} (\epsilon^i)^2$$

$$\underset{\text{Nx1}}{\begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix}} = \underset{\text{NxM}}{\begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}^1)^T \\ \hline \boldsymbol{\phi}(\mathbf{x}^2)^T \\ \hline \vdots \\ \hline \boldsymbol{\phi}(\mathbf{x}^N)^T \end{bmatrix}} \underset{\text{Mx1}}{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix}} + \underset{\text{Nx1}}{\begin{bmatrix} \epsilon^1 \\ \epsilon^2 \\ \vdots \\ \epsilon^N \end{bmatrix}}$$

$$\boldsymbol{\phi}(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^M$$

LS Solution for Linear Regression

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$$

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

$$\mathbf{X} = \begin{bmatrix} \frac{(\mathbf{x}^1)^T}{(\mathbf{x}^2)^T} \\ \hline \vdots \\ \hline (\mathbf{x}^N)^T \end{bmatrix}$$

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

LS Solution for Generalized Linear Regression

$$\mathbf{y} = \mathbf{\Phi w} + \boldsymbol{\epsilon}$$

$$\mathbf{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}^1)^T \\ \hline \boldsymbol{\phi}(\mathbf{x}^2)^T \\ \hline \vdots \\ \hline \boldsymbol{\phi}(\mathbf{x}^N)^T \end{bmatrix}$$

LS Solution for Generalized Linear Regression

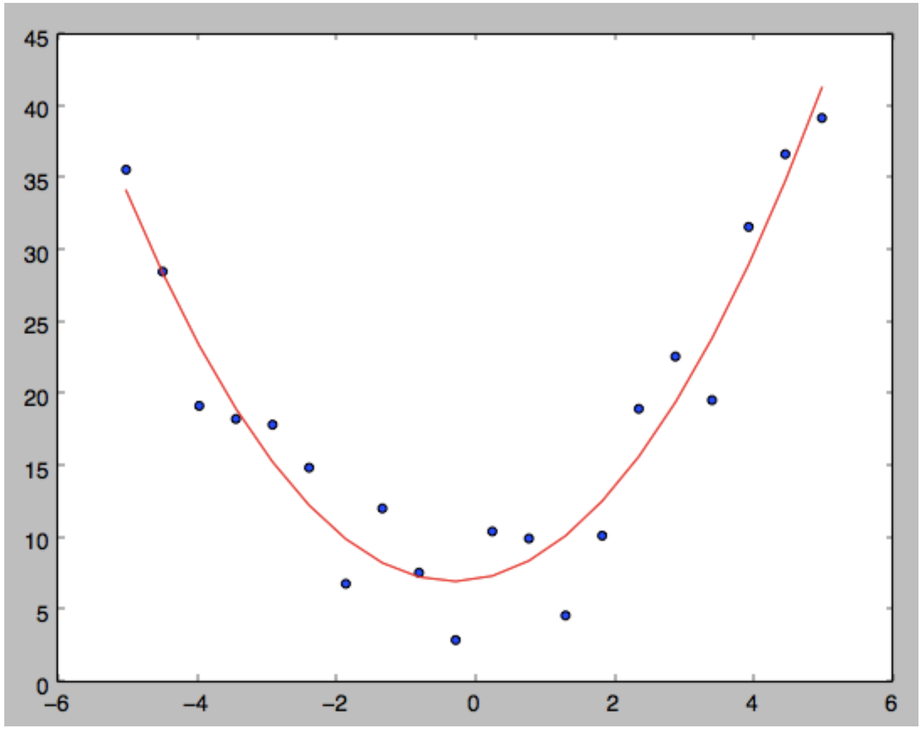
$$\mathbf{y} = \mathbf{\Phi w} + \boldsymbol{\epsilon}$$

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

$$\mathbf{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}^1)^T \\ \hline \boldsymbol{\phi}(\mathbf{x}^2)^T \\ \hline \vdots \\ \hline \boldsymbol{\phi}(\mathbf{x}^N)^T \end{bmatrix}$$

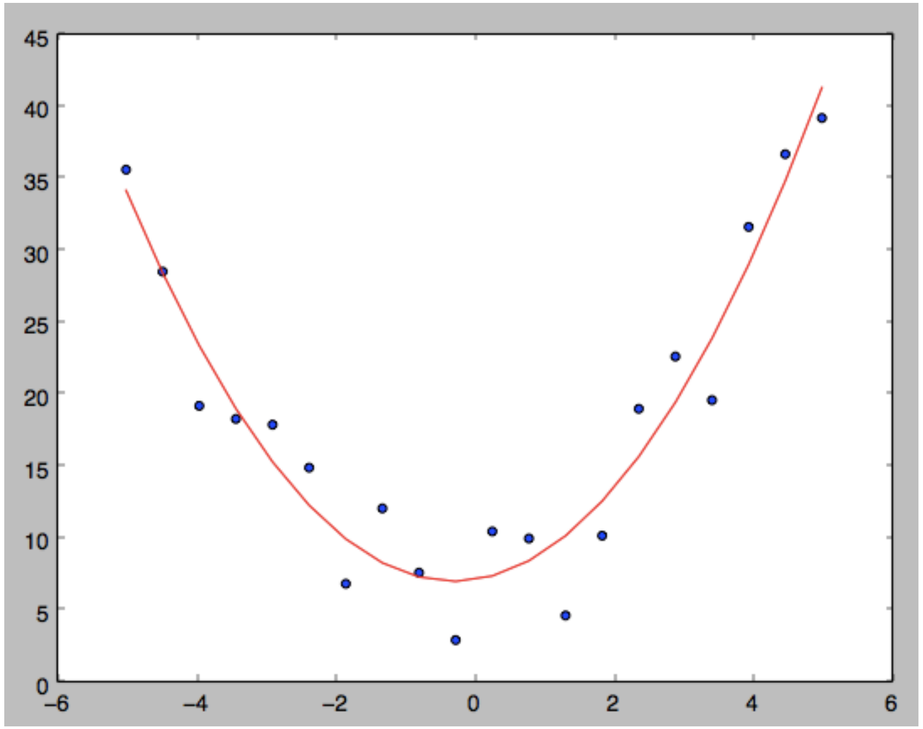
$$\mathbf{w}^* = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi y}$$

Code Example




```
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 3
w = 2*rand(3,1) # w[0] is random constant term (offset from origin), w[1] is random linear term, w[2] is random quadratic term
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + w[2]*x**2 + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1), (x**2).reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x + w_est[2]*x**2

# visualize the fitted model
pyplot.scatter(x, y)
pyplot.plot(x, y_est, color='red')
pyplot.show()
```

Code Example

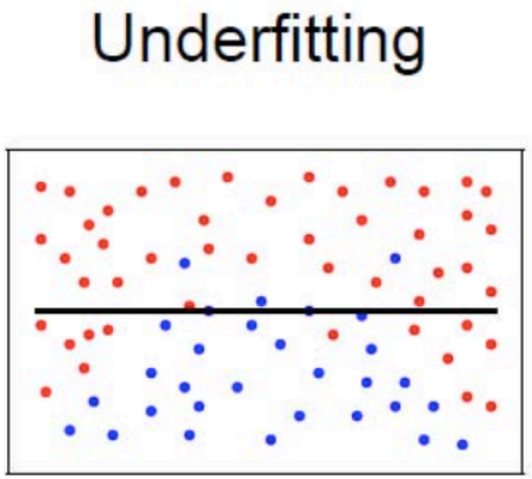



```
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 3
w = 2*rand(3,1) # w[0] is random constant term (offset from origin), w[1] is random linear term, w[2] is random quadratic term
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + w[2]*x**2 + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1), (x**2).reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x + w_est[2]*x**2

# visualize the fitted model
pyplot.scatter(x, y)
pyplot.plot(x, y_est, color='red')
pyplot.show()
```

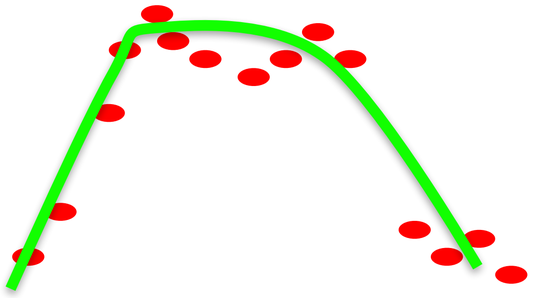
$$\mathbf{w}^* = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}\mathbf{y}$$

Underfitting vs. Overfitting

classification



regression

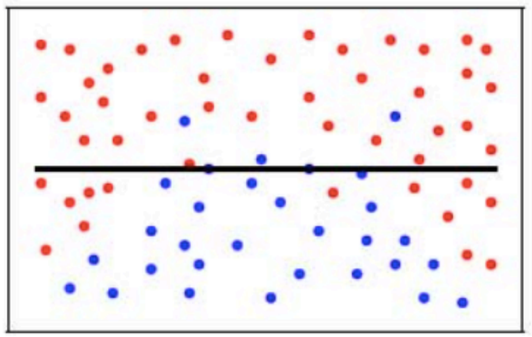


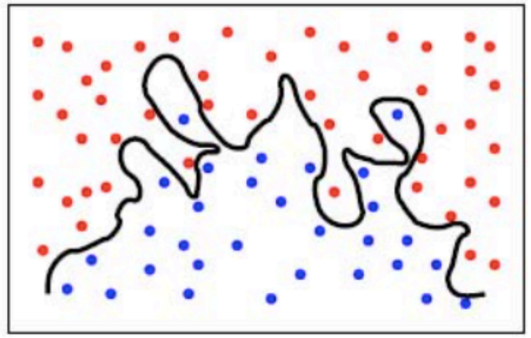
Underfitting vs. Overfitting

Underfitting

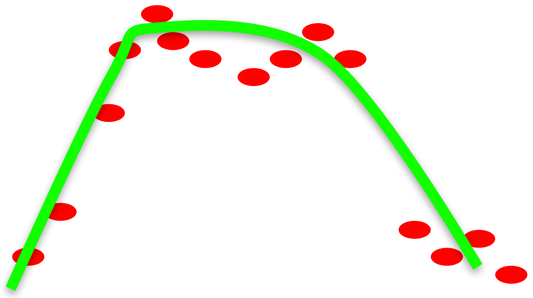
Overfitting

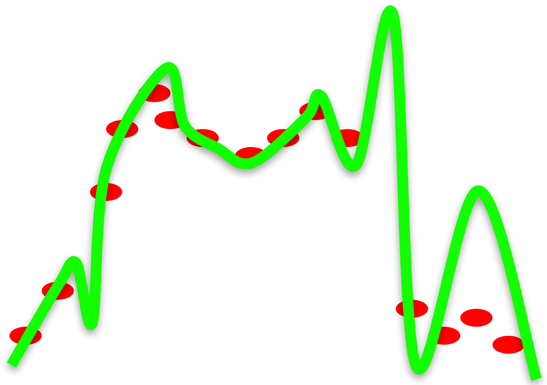
classification



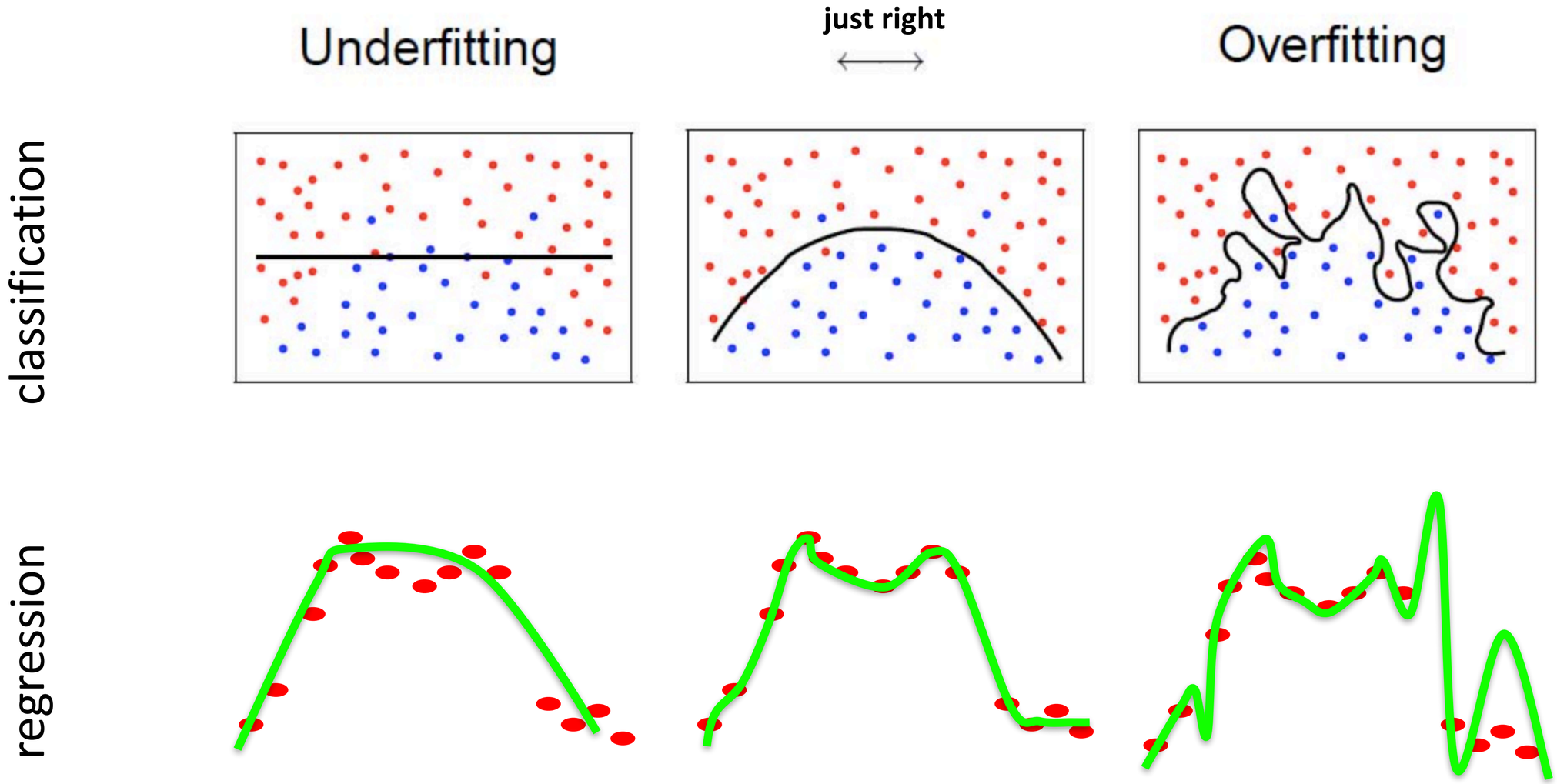


regression





Underfitting vs. Overfitting



Tuning Model's Complexity

Tuning Model's Complexity

A ***flexible model*** approximates the target function well in the training set

but can "**overtrain**" and have poor performance on the test set ("variance").

Tuning Model's Complexity

A ***flexible model*** approximates the target function well in the training set

but can "**overtrain**" and have poor performance on the test set ("variance").

A ***rigid model***'s performance is more predictable in the test set

but the model may not be good even on the training set ("bias").

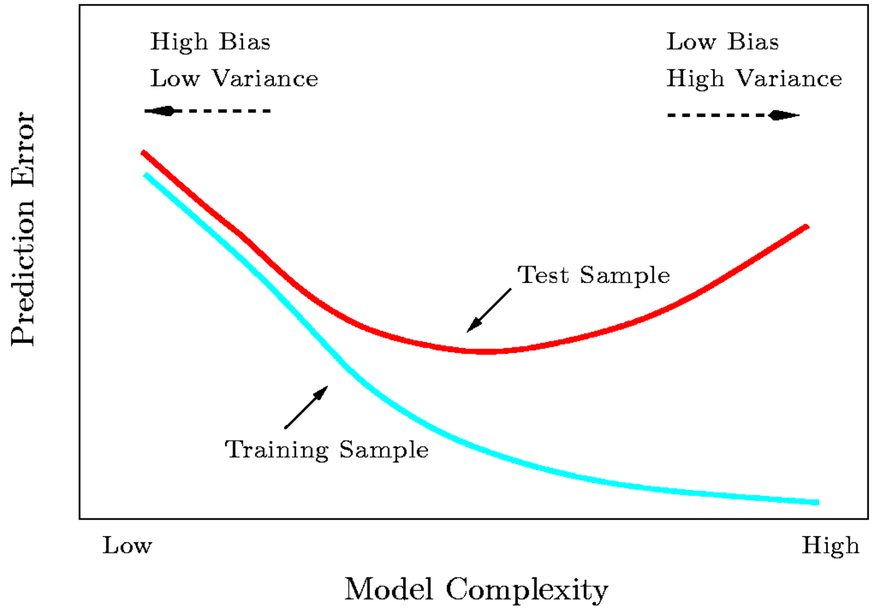
Tuning Model's Complexity

A ***flexible model*** approximates the target function well in the training set

but can "**overtrain**" and have poor performance on the test set ("variance").

A ***rigid model***'s performance is more predictable in the test set

but the model may not be good even on the training set ("bias").



Regularized Linear Regression

$$\boldsymbol{\epsilon} = \mathbf{y} - \boldsymbol{\Phi}\mathbf{w}$$

residual vector

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

linear regression: minimize model error

Regularized Linear Regression

$$\boldsymbol{\epsilon} = \mathbf{y} - \boldsymbol{\Phi}\mathbf{w}$$ residual vector

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$ linear regression: minimize model error

Complexity term: (regularizer) $$R(\mathbf{w}) \doteq \|\mathbf{w}\|_2^2 = \mathbf{w}^T \mathbf{w}$$

Regularized Linear Regression

$$\boldsymbol{\epsilon} = \mathbf{y} - \boldsymbol{\Phi}\mathbf{w}$$ residual vector

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$ linear regression: minimize model error

Complexity term: (regularizer) $$R(\mathbf{w}) \doteq \|\mathbf{w}\|_2^2 = \mathbf{w}^T \mathbf{w}$$

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + \lambda \mathbf{w}^T \mathbf{w}$$

Regularized Linear Regression

$$\boldsymbol{\epsilon} = \mathbf{y} - \boldsymbol{\Phi}\mathbf{w}$$ residual vector

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$$ linear regression: minimize model error

Complexity term: (regularizer) $$R(\mathbf{w}) \doteq \|\mathbf{w}\|_2^2 = \mathbf{w}^T\mathbf{w}$$

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} + \lambda\mathbf{w}^T\mathbf{w}$$

"data fidelity"

Regularized Linear Regression

$$\boldsymbol{\epsilon} = \mathbf{y} - \boldsymbol{\Phi}\mathbf{w}$$ residual vector

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$$ linear regression: minimize model error

Complexity term: (regularizer) $$R(\mathbf{w}) \doteq \|\mathbf{w}\|_2^2 = \mathbf{w}^T\mathbf{w}$$

$$L(\mathbf{w}) = \underbrace{\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}}_{\text{“data fidelity”}} + \underbrace{\lambda\mathbf{w}^T\mathbf{w}}_{\text{complexity}}$$

Regularized Linear Regression

$$\boldsymbol{\epsilon} = \mathbf{y} - \boldsymbol{\Phi}\mathbf{w}$$ residual vector

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$$ linear regression: minimize model error

Complexity term: (regularizer) $$R(\mathbf{w}) \doteq \|\mathbf{w}\|_2^2 = \mathbf{w}^T\mathbf{w}$$

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} + \lambda\mathbf{w}^T\mathbf{w}$$

"data fidelity" complexity

minimum remains to be determined

Regularized Linear Regression

$$\boldsymbol{\epsilon} = \mathbf{y} - \boldsymbol{\Phi}\mathbf{w}$$ residual vector

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$$ linear regression: minimize model error

Complexity term: (regularizer) $$R(\mathbf{w}) \doteq \|\mathbf{w}\|_2^2 = \mathbf{w}^T\mathbf{w}$$

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} + \lambda\mathbf{w}^T\mathbf{w}$$

"data fidelity" complexity

minimum remains to be determined

scalar, remains to be determined

Least Squares Solution

Least Squares Solution

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

Least Squares Solution

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$
$$= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Least Squares Solution

$$
\begin{aligned}
L(\mathbf{w}) &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \\
&= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}
\end{aligned}
$$

Least Squares Solution

$$
\begin{aligned}
L(\mathbf{w}) &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \\
&= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}
\end{aligned}
$$

Condition for minimum:

$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$

Least Squares Solution

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$
$$= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$
$$= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}$$

Condition for minimum:

$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$
$$-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w}^* = \mathbf{0}$$

Least Squares Solution

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$
$$= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$
$$= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}$$

Condition for minimum:

$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$
$$-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w}^* = \mathbf{0}$$
$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Ridge regression: L2-regularized Linear Regression

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + \lambda \mathbf{w}^T \mathbf{w}$$

$$= \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} + \lambda \mathbf{w}^T \mathbf{I}\mathbf{w}$$

as before, for linear regression

identity matrix

$$= \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{w}^T \left(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}\right) \mathbf{w}$$

Condition for minimum:

$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$

$$-2\mathbf{X}^T \mathbf{y} + 2(\mathbf{X}^T \mathbf{X} + \lambda I)\mathbf{w}^* = \mathbf{0}$$

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

Ridge regression: L2-regularized Linear Regression

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + \lambda \mathbf{w}^T \mathbf{w}$$

$$= \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} + \lambda \mathbf{w}^T \mathbf{I}\mathbf{w}$$

as before, for linear regression

identity matrix

$$= \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{w}^T \left(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}\right) \mathbf{w}$$

Condition for minimum:

$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$

$$-2\mathbf{X}^T \mathbf{y} + 2(\mathbf{X}^T \mathbf{X} + \lambda I)\mathbf{w}^* = \mathbf{0}$$

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

Ridge regression: L2-regularized Linear Regression

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} + \lambda \mathbf{w}^T\mathbf{w}$$

$$= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}^T\mathbf{I}\mathbf{w}$$

as before, for linear regression

identity matrix

$$= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)\mathbf{w}$$

Condition for minimum:

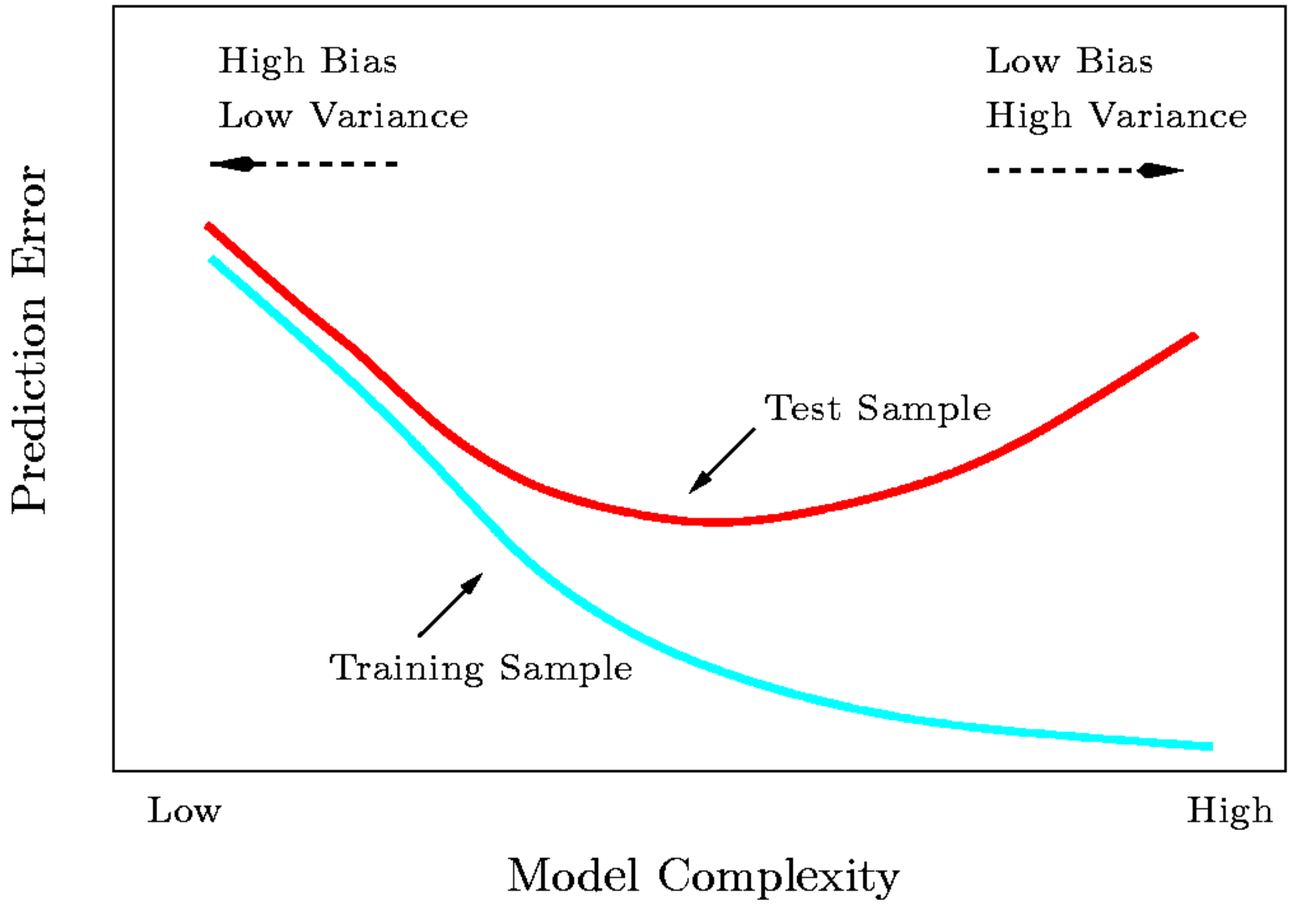
$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$

$$-2\mathbf{X}^T\mathbf{y} + 2(\mathbf{X}^T\mathbf{X} + \lambda I)\mathbf{w}^* = \mathbf{0}$$

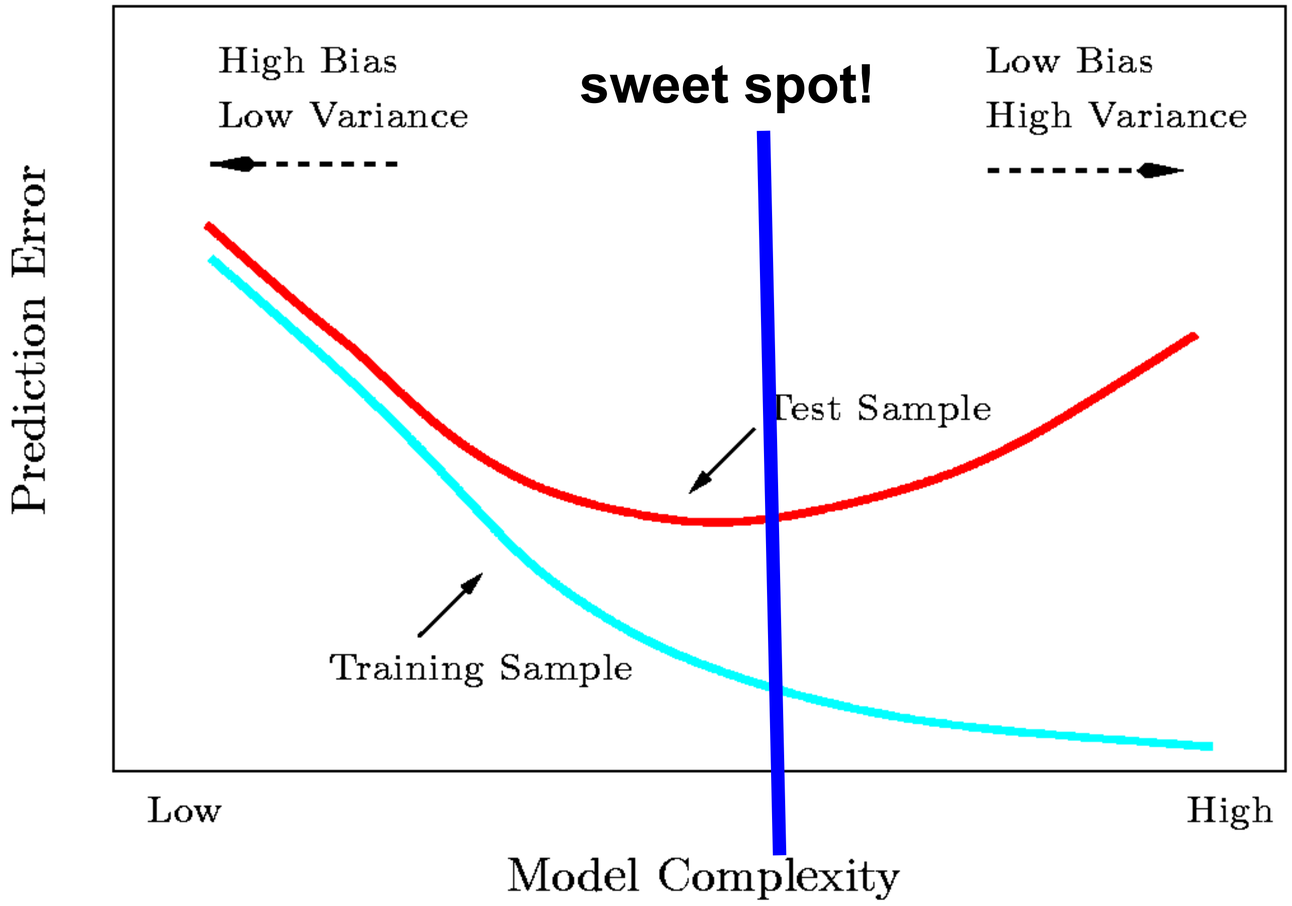
$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

λ: "hyperparameter"

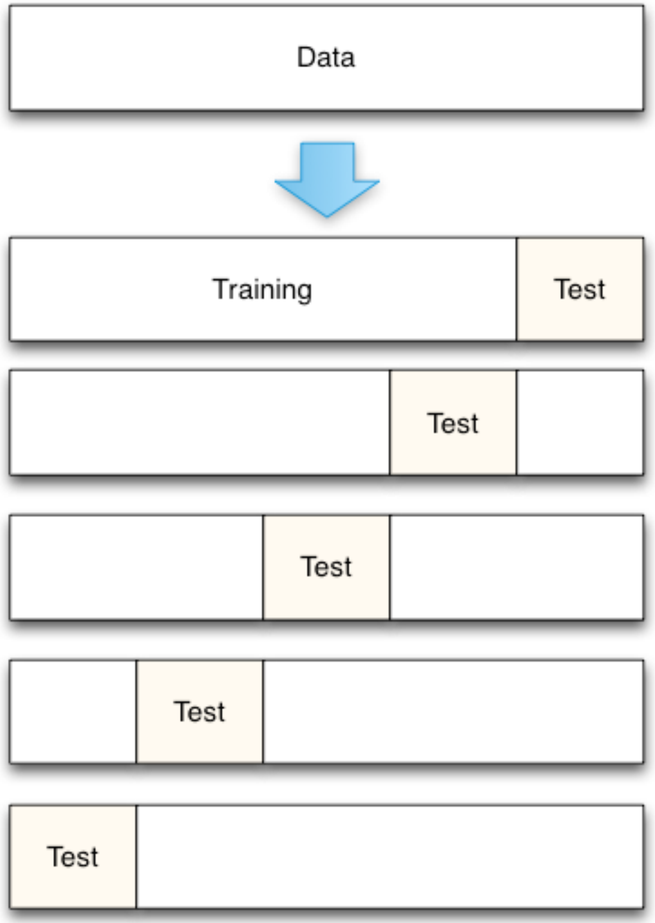
Bias-Variance Tradeoff (function of λ)



Bias-Variance Tradeoff (function of λ)

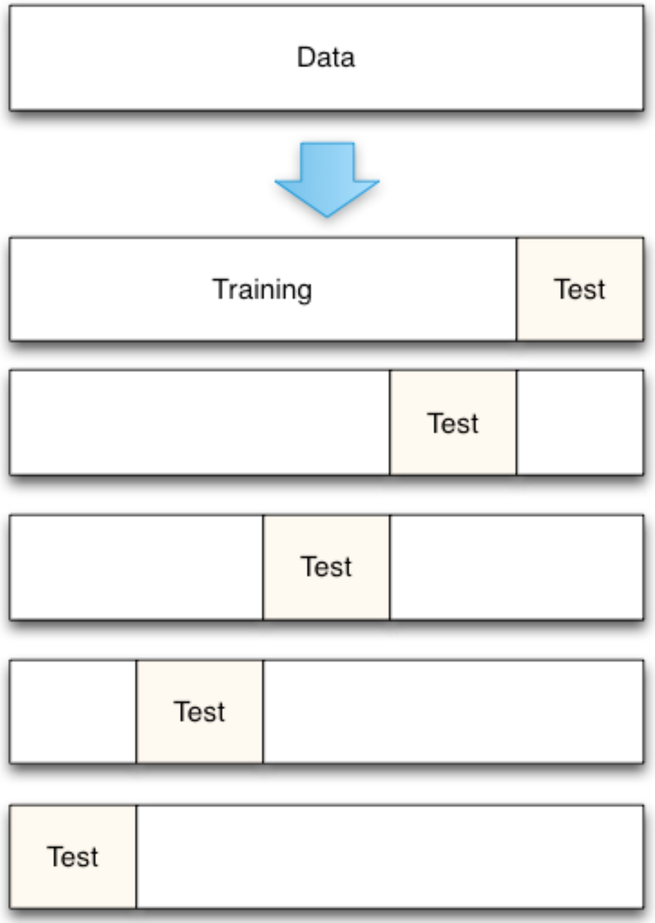


Selecting λ with Cross-validation



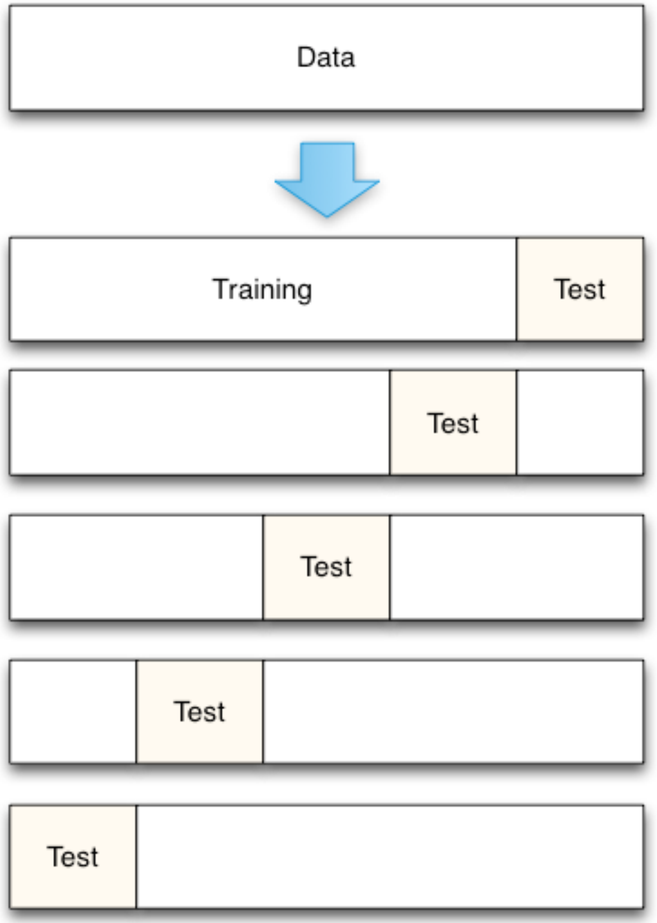
Selecting λ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error



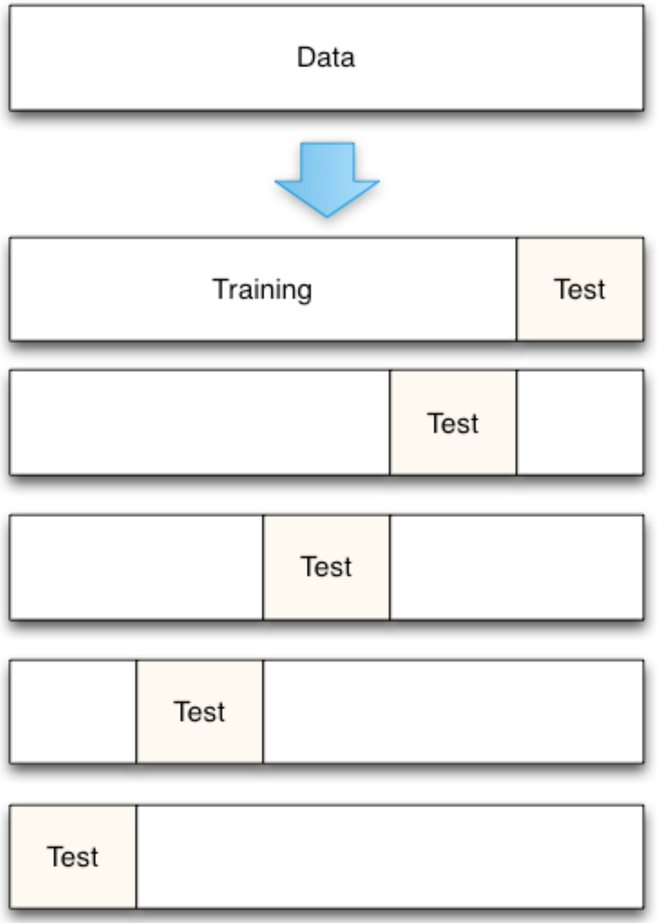
Selecting λ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error
- K-fold cross validation:
  - K splits, average K errors



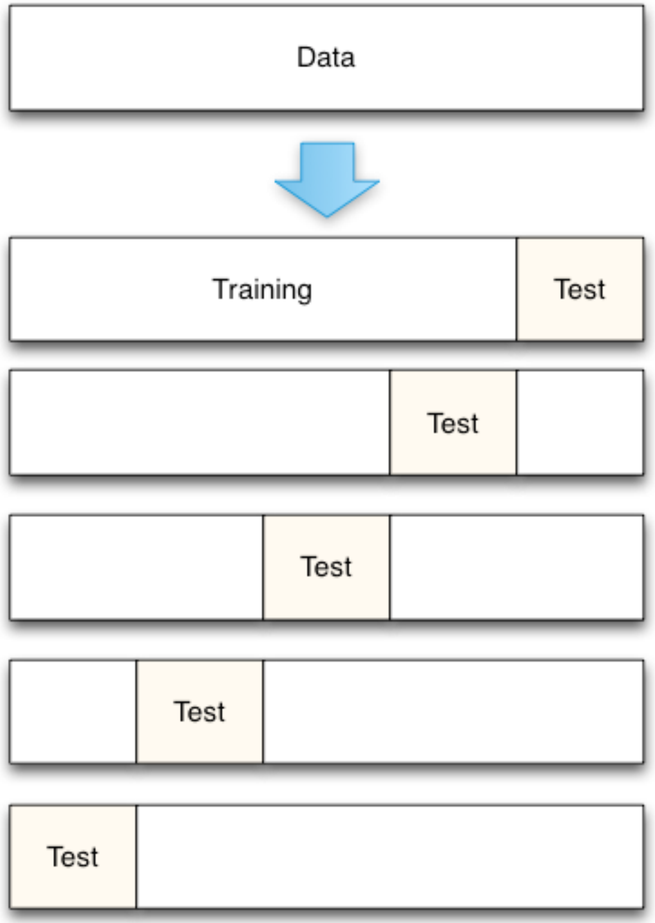
Selecting λ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error
- K-fold cross validation:
  - K splits, average K errors



Selecting λ with Cross-validation

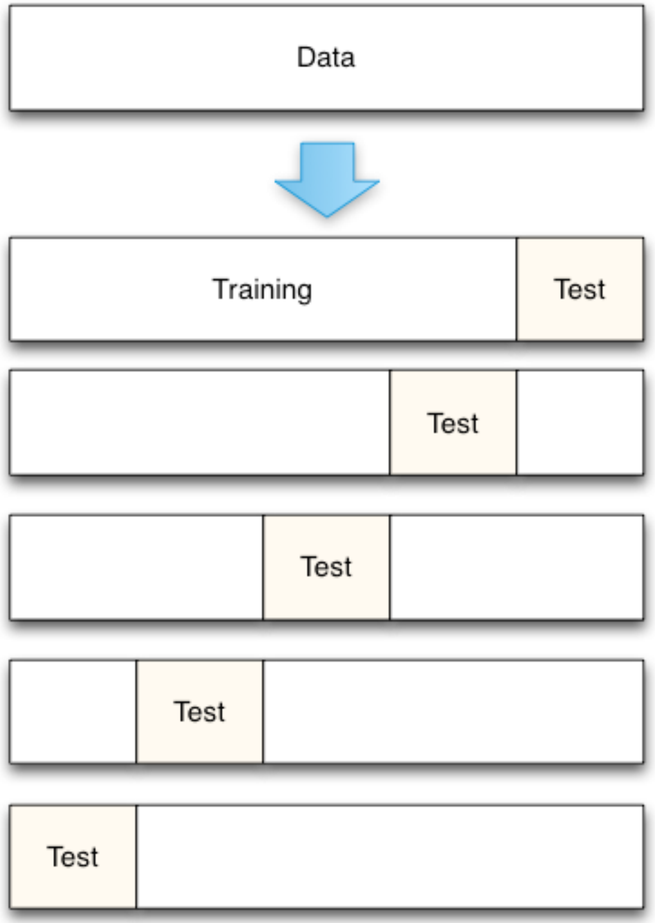
- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error
- K-fold cross validation:
  - K splits, average K errors
- Use cross-validation for different values of λ
  - pick value that minimizes cross-validation error



Selecting λ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error
- K-fold cross validation:
  - K splits, average K errors
- Use cross-validation for different values of λ
  - pick value that minimizes cross-validation error

Least glorious, most effective of all methods



Form of posterior distribution

Bernoulli-type conditional distribution

$$\left.\begin{aligned} P(Y=1|X=\mathbf{x};\mathbf{w}) &= f(\mathbf{x},\mathbf{w}) \\ P(Y=0|X=\mathbf{x};\mathbf{w}) &= 1-f(\mathbf{x},\mathbf{w}) \end{aligned}\right\} \rightarrow$$

Particular choice of form of f:

Form of posterior distribution

Bernoulli-type conditional distribution

$$\left.\begin{aligned} P(Y=1|X=\mathbf{x};\mathbf{w}) &= f(\mathbf{x},\mathbf{w}) \\ P(Y=0|X=\mathbf{x};\mathbf{w}) &= 1-f(\mathbf{x},\mathbf{w}) \end{aligned}\right\} \rightarrow$$

Particular choice of form of f:

$$P(Y=1|X=\mathbf{x};\mathbf{w}) = g(\mathbf{w}^T\mathbf{x})$$

Form of posterior distribution

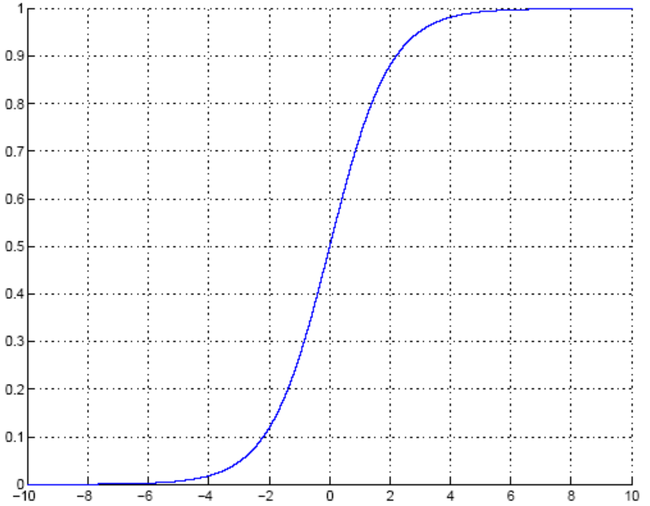
Bernoulli-type conditional distribution

$$\left.\begin{aligned} P(Y=1|X=\mathbf{x};\mathbf{w}) &= f(\mathbf{x},\mathbf{w}) \\ P(Y=0|X=\mathbf{x};\mathbf{w}) &= 1-f(\mathbf{x},\mathbf{w}) \end{aligned}\right\} \rightarrow$$

Particular choice of form of f:

$$P(Y=1|X=\mathbf{x};\mathbf{w}) = g(\mathbf{w}^T\mathbf{x})$$

Sigmoidal: $$g(\alpha) = \frac{1}{1+\exp(-a)}$$



Form of posterior distribution

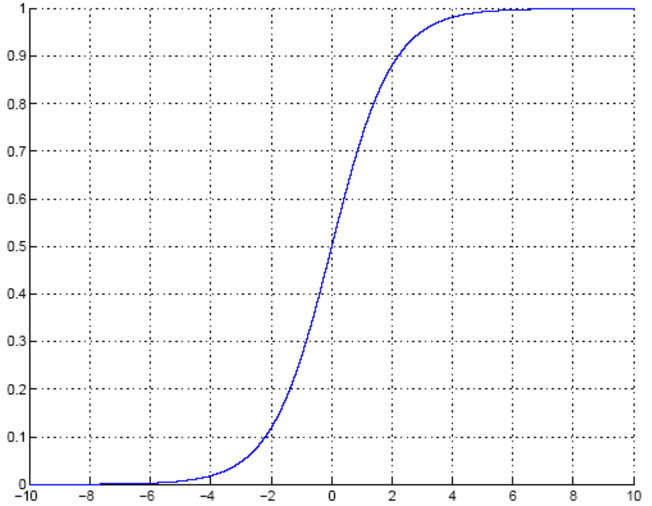
Bernoulli-type conditional distribution

$$\left.\begin{aligned} P(Y=1|X=\mathbf{x};\mathbf{w}) &= f(\mathbf{x},\mathbf{w}) \\ P(Y=0|X=\mathbf{x};\mathbf{w}) &= 1-f(\mathbf{x},\mathbf{w}) \end{aligned}\right\} \rightarrow$$

Particular choice of form of f:

$$P(Y=1|X=\mathbf{x};\mathbf{w}) = g(\mathbf{w}^T\mathbf{x})$$

Sigmoidal: $$g(\alpha) = \frac{1}{1+\exp(-a)}$$



"squashing function":

$$-\infty \rightarrow 0$$
$$+\infty \rightarrow 1$$

Form of posterior distribution

Bernoulli-type conditional distribution

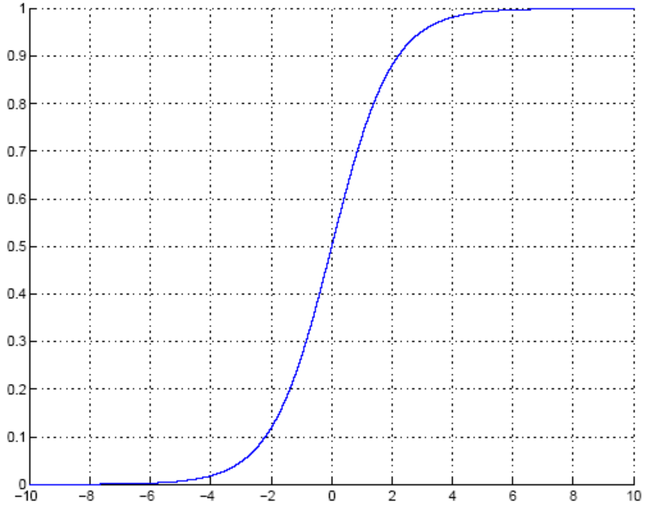
$$\left.\begin{aligned} P(Y=1|X=\mathbf{x};\mathbf{w}) &= f(\mathbf{x},\mathbf{w}) \\ P(Y=0|X=\mathbf{x};\mathbf{w}) &= 1-f(\mathbf{x},\mathbf{w}) \end{aligned}\right\} \rightarrow$$

$$P(Y=y|X=\mathbf{x};\mathbf{w}) = f(\mathbf{x},\mathbf{w})^{y}(1-f(\mathbf{x},\mathbf{w}))^{1-y}$$

Particular choice of form of f:

$$P(Y=1|X=\mathbf{x};\mathbf{w}) = g(\mathbf{w}^{T}\mathbf{x})$$

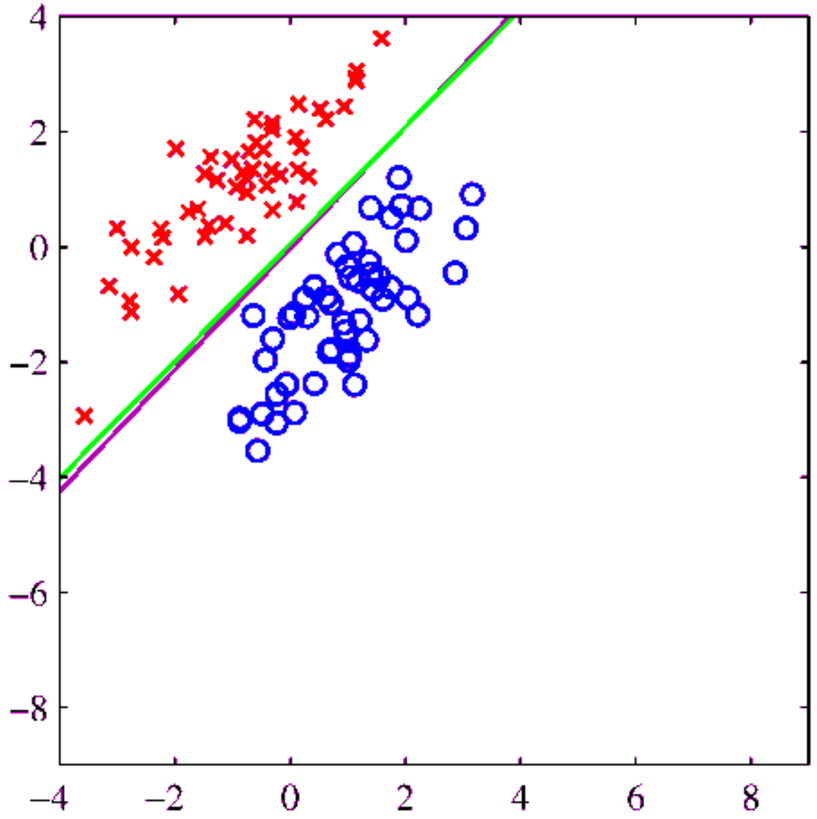
Sigmoidal: $g(\alpha) = \dfrac{1}{1+\exp(-a)}$



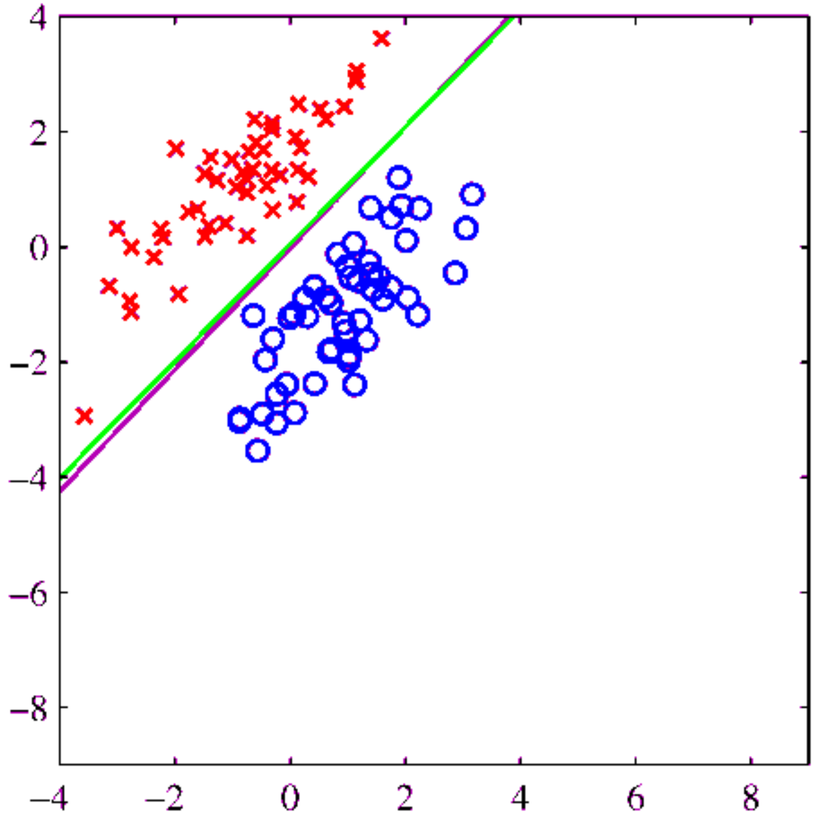
"squashing function":

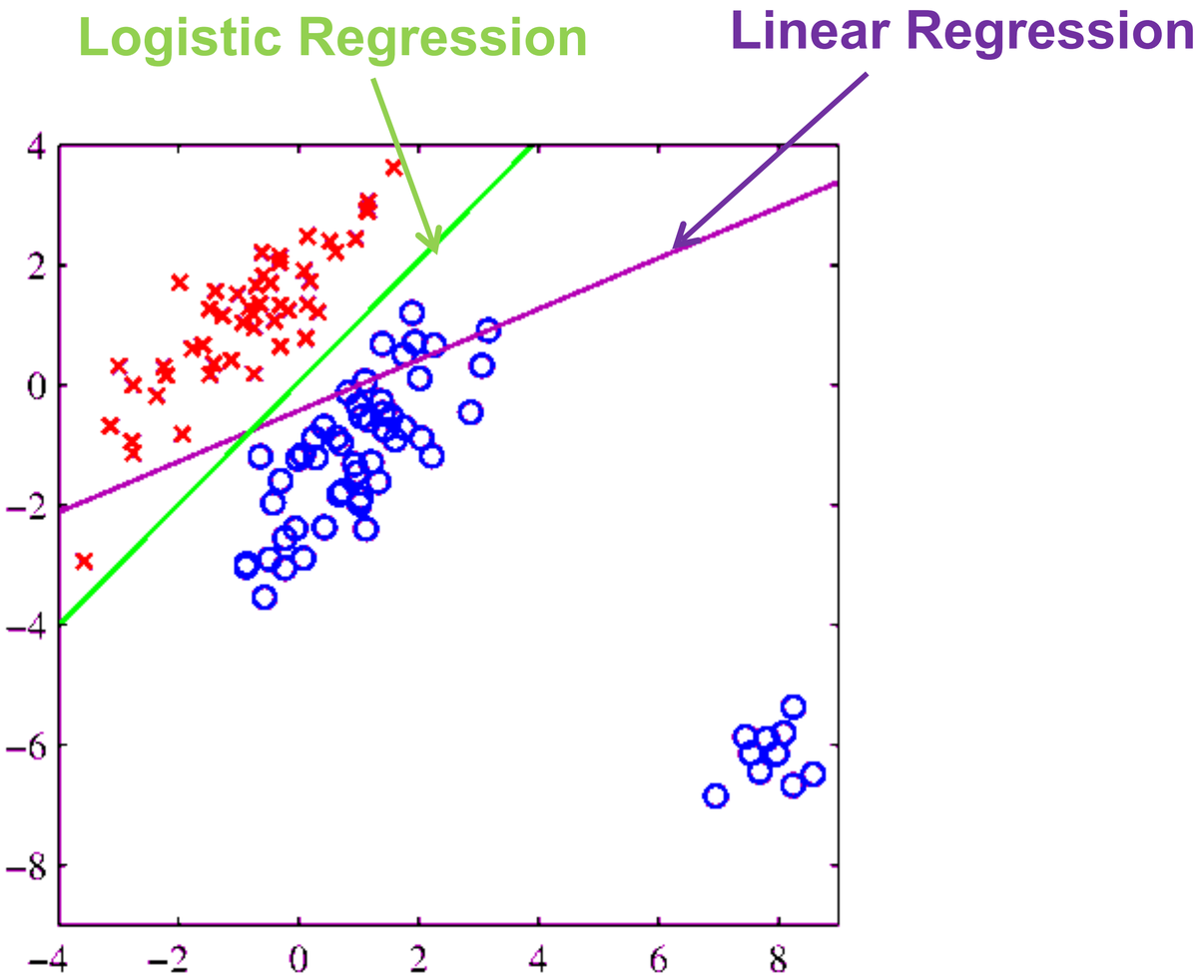
$$-\infty \rightarrow 0$$
$$+\infty \rightarrow 1$$

Logistic vs Linear Regression



Logistic vs Linear Regression





From Two to Many

- How about multi-class classification?

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:

$$\mathbf{y}^i = (0, 0, 1, 0)$$

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:

$$\mathbf{y}^i = (0, 0, 1, 0)$$

Matrix notation:

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:

$$\mathbf{y}^i = (0, 0, 1, 0)$$

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \hline \vdots \\ \hline \mathbf{y}^N \end{bmatrix}$$

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:

$$\mathbf{y}^i = (0, 0, 1, 0)$$

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \hline \vdots \\ \hline \mathbf{y}^N \end{bmatrix} = \left[\begin{array}{c|c|c} \mathbf{y}_1 & \dots & \mathbf{y}_C \end{array} \right]$$

where $\mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:

$$\mathbf{y}^i = (0, 0, 1, 0)$$

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \hline \vdots \\ \hline \mathbf{y}^N \end{bmatrix} = \left[\begin{array}{c|c|c} \mathbf{y}_1 & \dots & \mathbf{y}_C \end{array} \right]$$

where $\mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$

$$\mathbf{W} = \left[\begin{array}{c|c|c} \mathbf{w}_1 & \dots & \mathbf{w}_C \end{array} \right]$$

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:

$$\mathbf{y}^i = (0, 0, 1, 0)$$

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \hline \vdots \\ \hline \mathbf{y}^N \end{bmatrix} = \left[\begin{array}{c|c|c} \mathbf{y}_1 & \dots & \mathbf{y}_C \end{array} \right] \quad \text{where} \quad \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

$$\mathbf{W} = \left[\begin{array}{c|c|c} \mathbf{w}_1 & \dots & \mathbf{w}_C \end{array} \right]$$

Loss function:

$$L(\mathbf{W}) = \sum_{c=1}^{C} (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)^T (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)$$

Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:

$$\mathbf{y}^i = (0, 0, 1, 0)$$

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \hline \vdots \\ \hline \mathbf{y}^N \end{bmatrix} = \left[\begin{array}{c|c|c} \mathbf{y}_1 & \dots & \mathbf{y}_C \end{array} \right] \quad \text{where} \quad \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

$$\mathbf{W} = \left[\begin{array}{c|c|c} \mathbf{w}_1 & \dots & \mathbf{w}_C \end{array} \right]$$

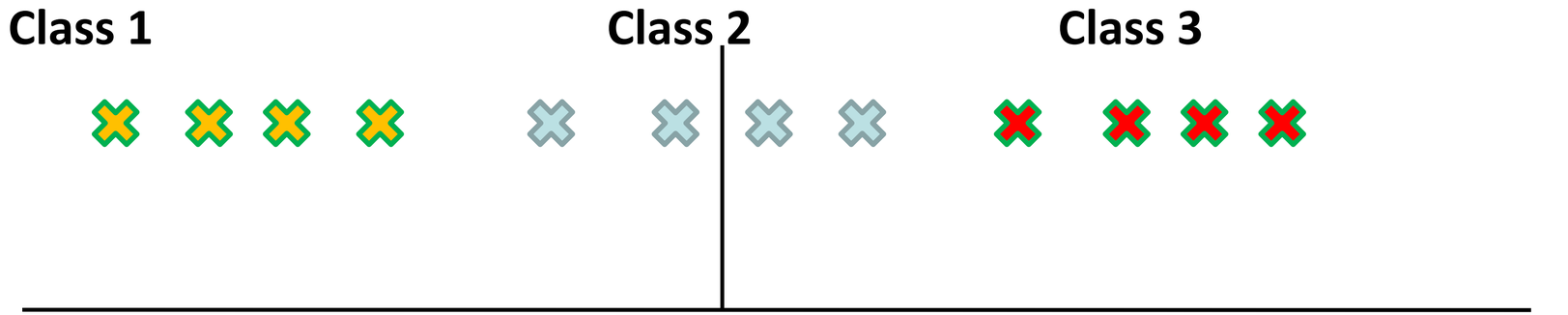
Loss function:

$$L(\mathbf{W}) = \sum_{c=1}^{C} (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)^T (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)$$

Least squares fit (decouples per class):

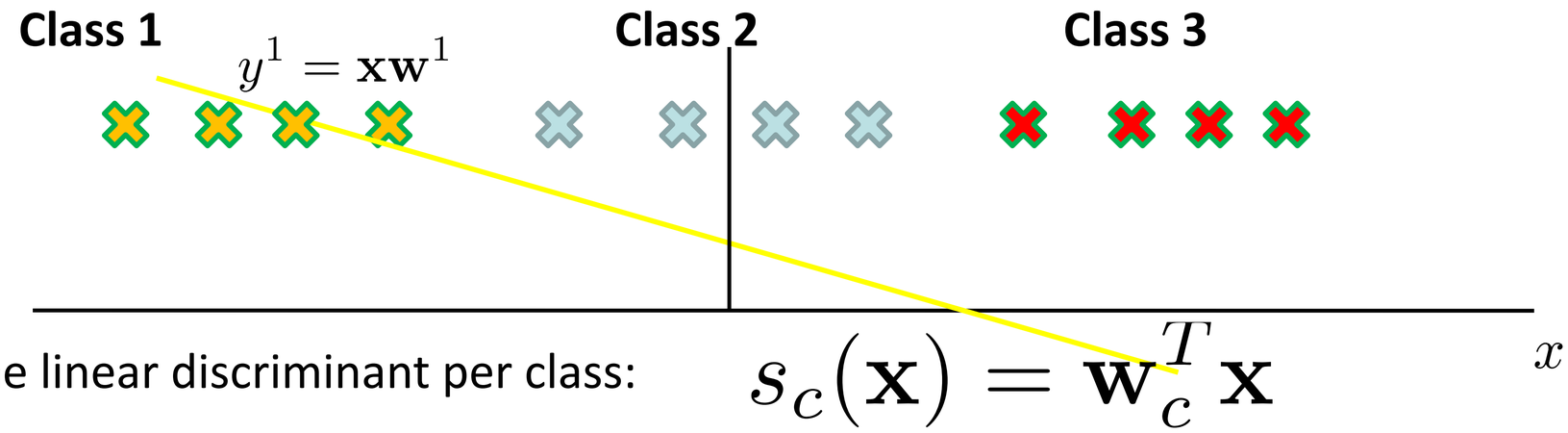
$$\mathbf{w}_c^* = (\mathbf{X}^T\mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}_c$$

Linear Regression Masking Problem



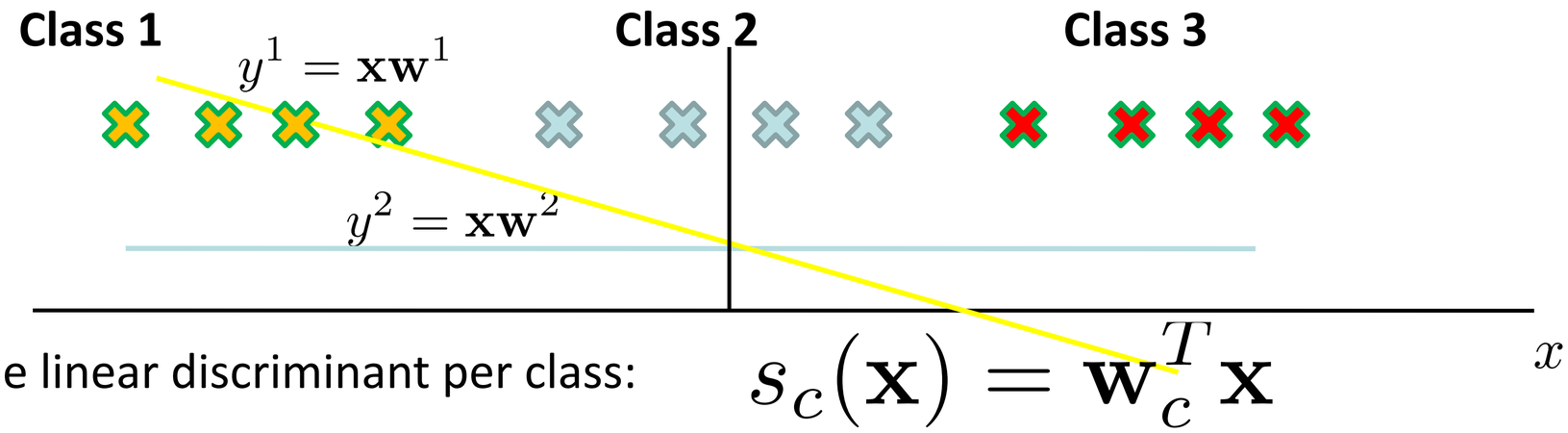
One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

Linear Regression Masking Problem



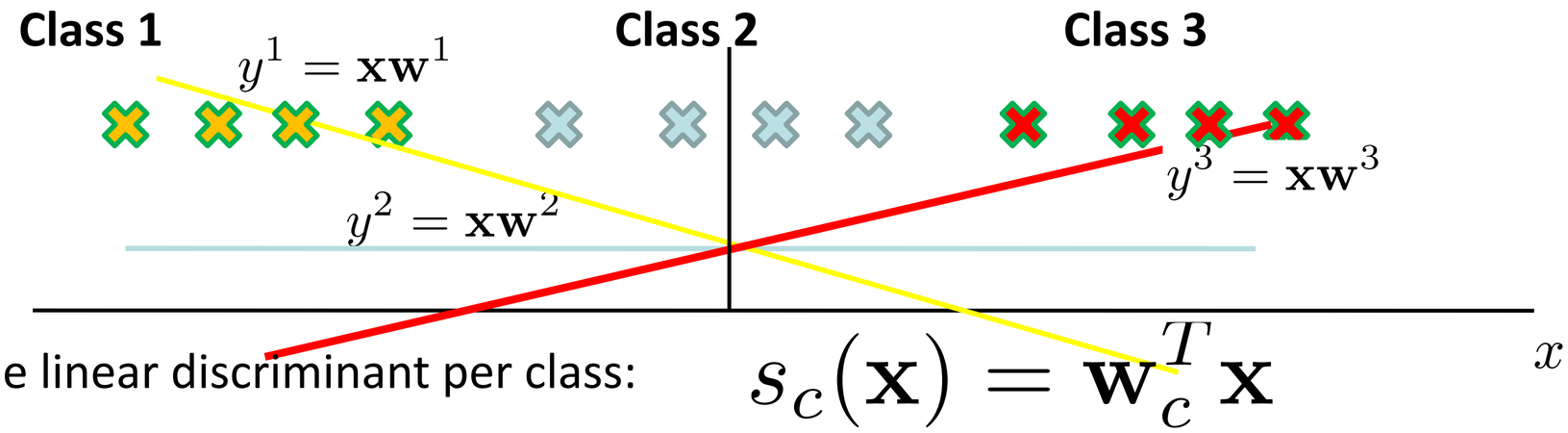
One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

Linear Regression Masking Problem



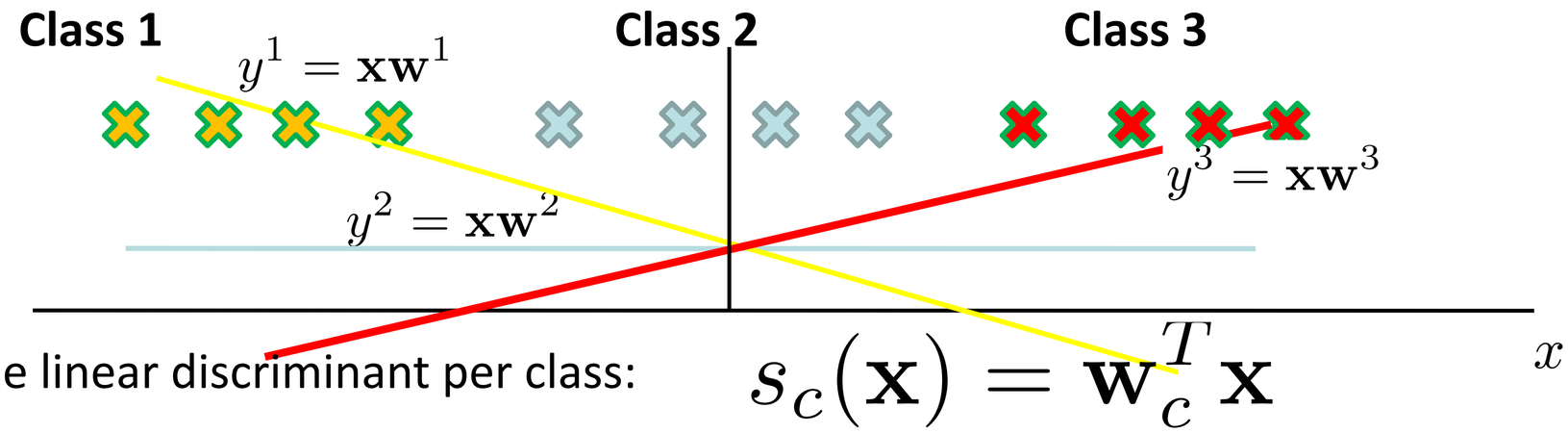
One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

Linear Regression Masking Problem



One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

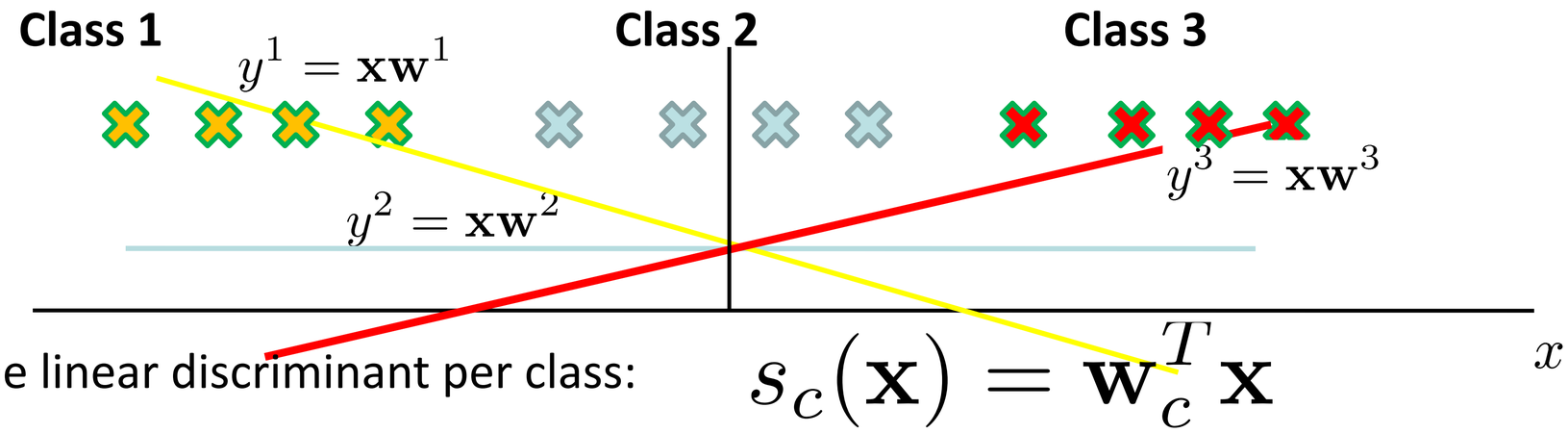
Linear Regression Masking Problem



One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

Nothing ever gets assigned to class 2!

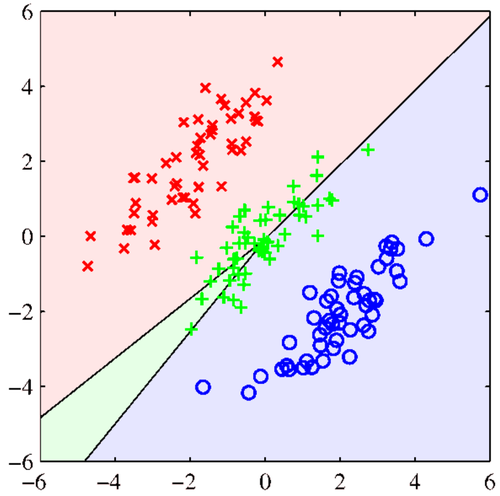
Linear Regression Masking Problem



One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

Nothing ever gets assigned to class 2!

2D version:



Multiple classes & Logistic regression

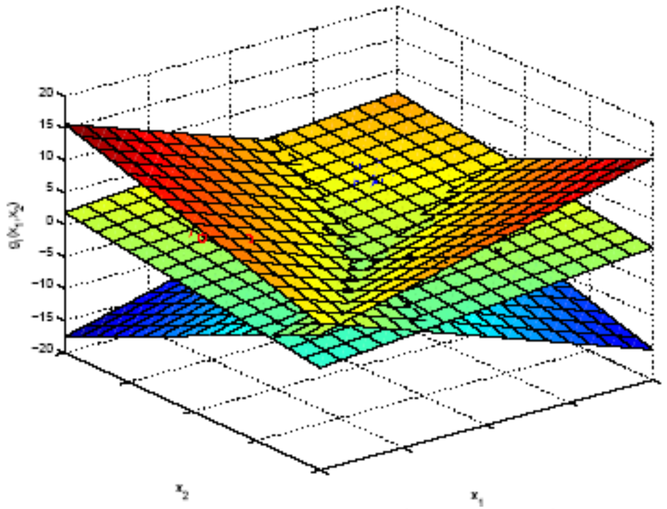
Soft maximum (softmax) of competing classes:

$$P(y = c|\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \mathbf{x})} \doteq g_c(\mathbf{x}, \mathbf{W})$$

Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

$$P(y = c|\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \mathbf{x})} \doteq g_c(\mathbf{x}, \mathbf{W})$$

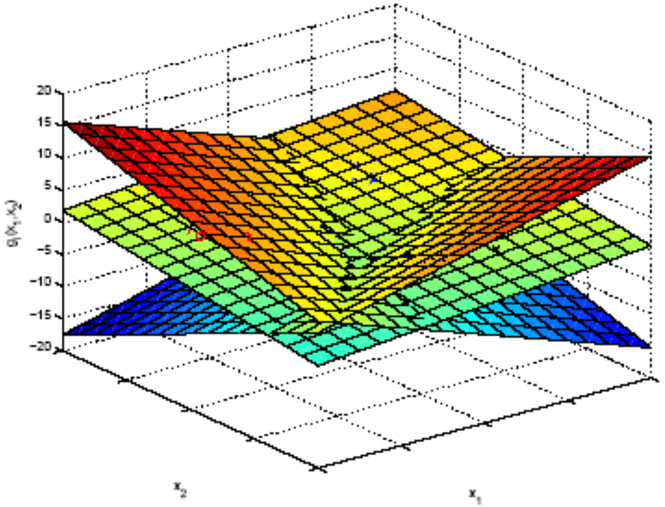


Discriminants (inputs)

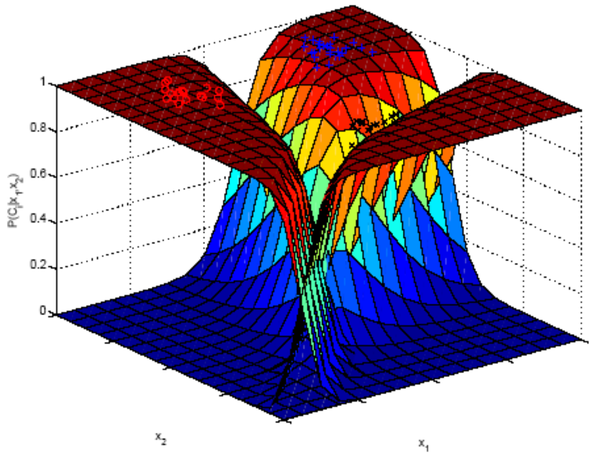
Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

$$P(y = c|\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \mathbf{x})} \doteq g_c(\mathbf{x}, \mathbf{W})$$



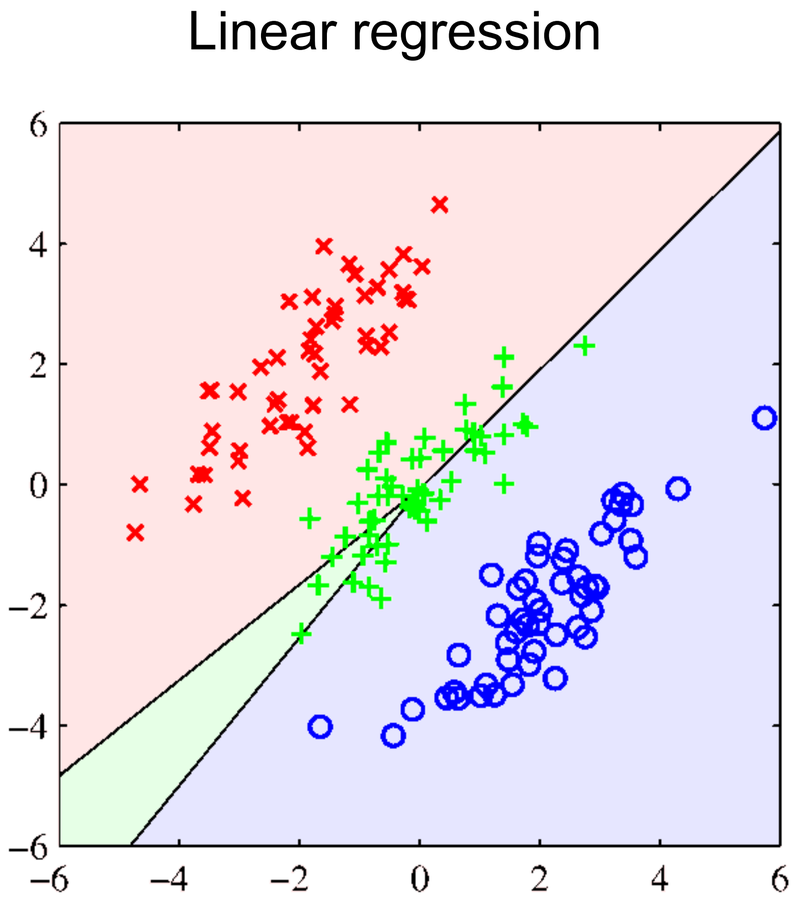
Discriminants (inputs)



Softmax (outputs)

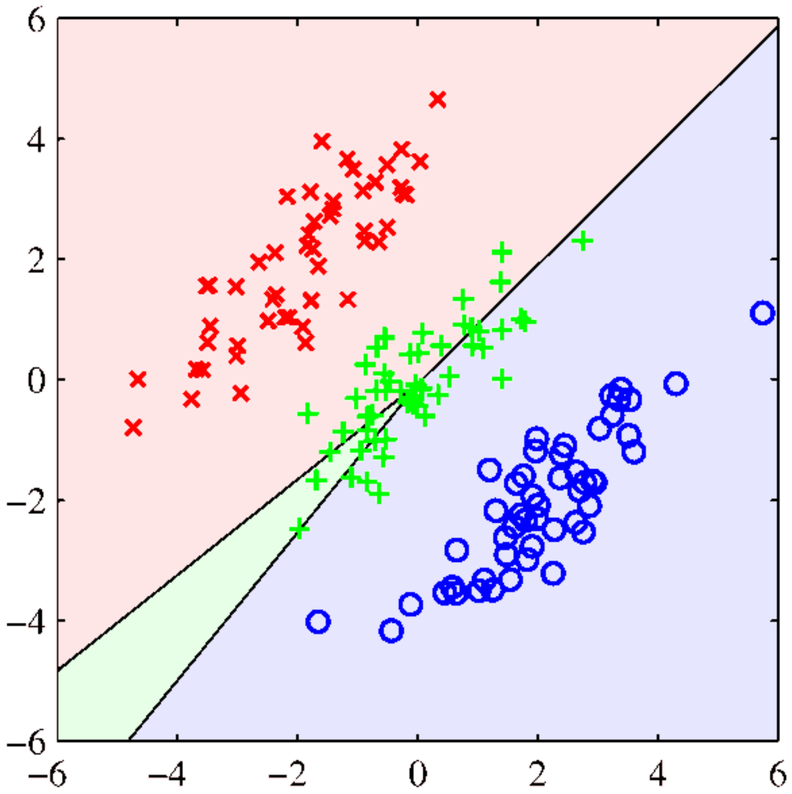
Logistic vs Linear Regression, n>2 classes

Logistic vs Linear Regression, n>2 classes

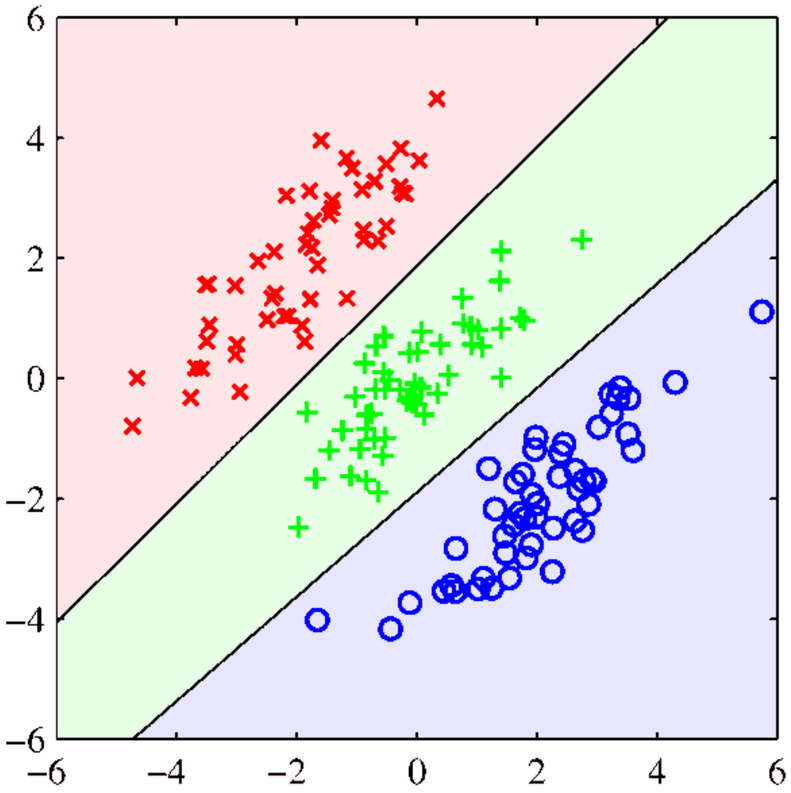


Logistic vs Linear Regression, n>2 classes

Linear regression

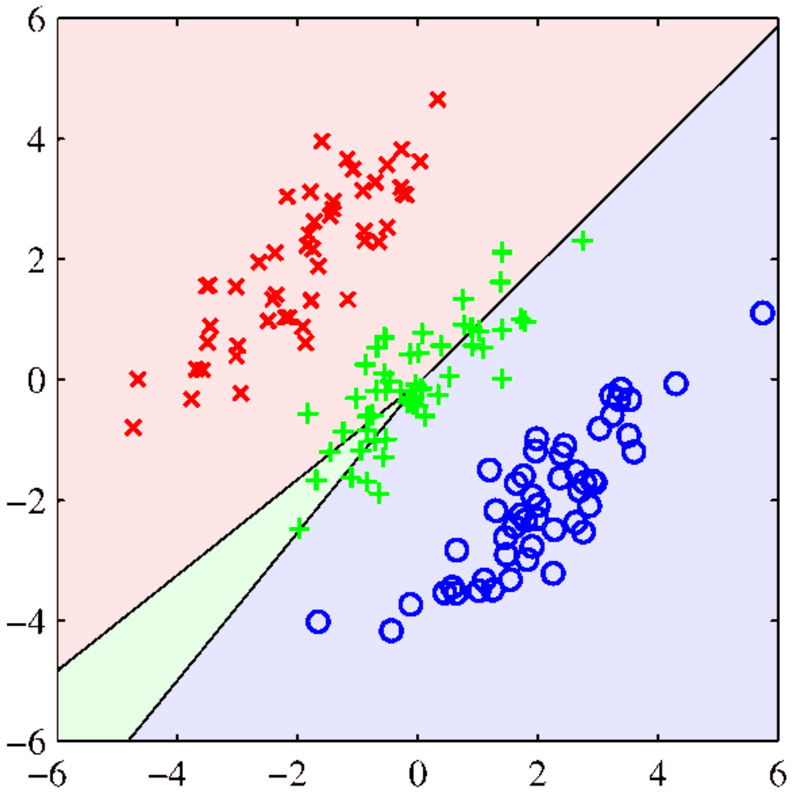


Logistic regression

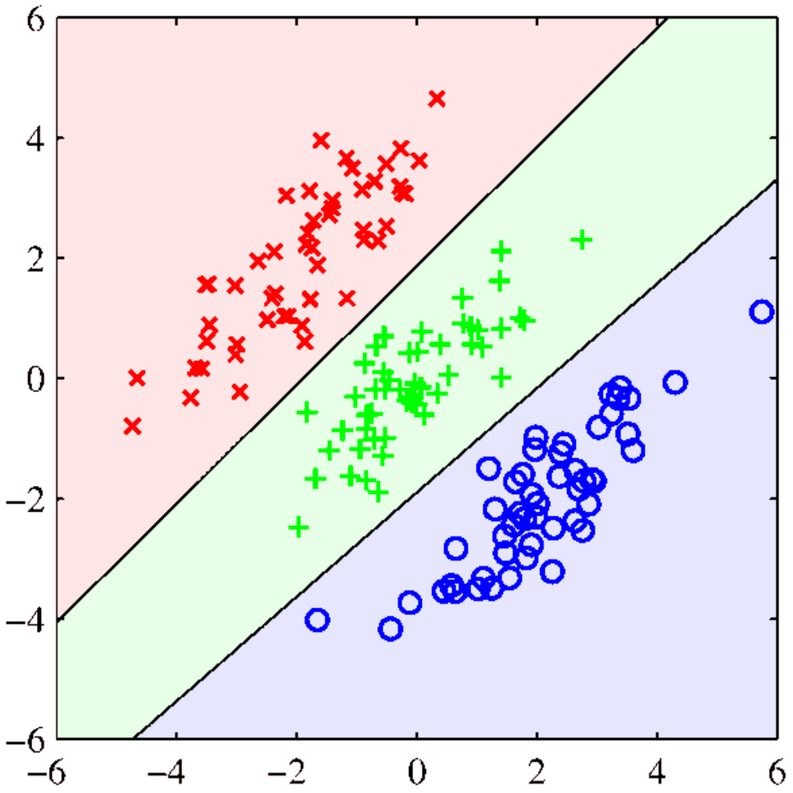


Logistic vs Linear Regression, n>2 classes

Linear regression



Logistic regression



Logistic regression does not exhibit the masking problem

LS Solution (in vector form)

LS Solution (in vector form)

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

LS Solution (in vector form)

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$
$$= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

LS Solution (in vector form)

$$
\begin{aligned}
L(\mathbf{w}) &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \\
&= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}
\end{aligned}
$$

LS Solution (in vector form)

$$
\begin{aligned}
L(\mathbf{w}) &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \\
&= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}
\end{aligned}
$$

Condition for minimum:

$$
\nabla L(\mathbf{w}^*) = \mathbf{0}
$$

LS Solution (in vector form)

$$
\begin{aligned}
L(\mathbf{w}) &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \\
&= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}
\end{aligned}
$$

Condition for minimum:

$$
\begin{aligned}
\nabla L(\mathbf{w}^*) &= \mathbf{0} \\
-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w}^* &= \mathbf{0}
\end{aligned}
$$

LS Solution (in vector form)

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$
$$= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$
$$= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}$$

Condition for minimum:

$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$
$$-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w}^* = \mathbf{0}$$
$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Gradient of Cross-entropy Loss

$$L(\mathbf{w}) = -\sum_{i=1}^{N} y^i \log g(\mathbf{w}^T \mathbf{x}^i) + (1 - y^i) \log(1 - g(\mathbf{w}^T \mathbf{x}^i))$$

Gradient of Cross-entropy Loss

$$L(\mathbf{w}) = -\sum_{i=1}^{N} y^i \log g(\mathbf{w}^T\mathbf{x}^i) + (1-y^i)\log(1-g(\mathbf{w}^T\mathbf{x}^i))$$

$$\frac{\partial L(\mathbf{w})}{\partial w_k} = -\sum_{i=1}^{N}\left[y^i \frac{1}{g(\mathbf{w}^T\mathbf{x}^i)}\frac{\partial g(\mathbf{w}^T\mathbf{x}^i)}{\partial w_k} + (1-y^i)\frac{1}{1-g(\mathbf{w}^T\mathbf{x}^i)}(-\frac{\partial g(\mathbf{w}^T\mathbf{x}^i)}{\partial w_k})\right]$$

using

$$g(x) = \frac{1}{1+\exp(-x)} \rightarrow \frac{dg}{dx} = g(x)(1-g(x))$$

$$= -\sum_{i=1}^{N}\left[y^i \frac{1}{g(\mathbf{w}^T\mathbf{x}^i)} - (1-y^i)\frac{1}{1-g(\mathbf{w}^T\mathbf{x}^i)}\right] g(\mathbf{w}^T\mathbf{x}^i)(1-g(\mathbf{w}^T\mathbf{x}^i))\frac{\partial \mathbf{w}^T\mathbf{x}^i}{\partial w_k}$$

$$= -\sum_{i=1}^{N}\left[y^i(1-g(\mathbf{w}^T\mathbf{x}^i)) - (1-y^i)g(\mathbf{w}^T\mathbf{x}^i)\right] x_k^i$$

$$= -\sum_{i=1}^{N}\left[y^i - g(\mathbf{w}^T\mathbf{x}^i)\right] \mathbf{x}_k^i$$

Gradient of Cross-entropy Loss

$$L(\mathbf{w}) = -\sum_{i=1}^{N} y^i \log g(\mathbf{w}^T\mathbf{x}^i) + (1-y^i)\log(1-g(\mathbf{w}^T\mathbf{x}^i))$$

$$\frac{\partial L(\mathbf{w})}{\partial w_k} = -\sum_{i=1}^{N}\left[y^i \frac{1}{g(\mathbf{w}^T\mathbf{x}^i)}\frac{\partial g(\mathbf{w}^T\mathbf{x}^i)}{\partial w_k} + (1-y^i)\frac{1}{1-g(\mathbf{w}^T\mathbf{x}^i)}(-\frac{\partial g(\mathbf{w}^T\mathbf{x}^i)}{\partial w_k})\right]$$

using

$$g(x) = \frac{1}{1+\exp(-x)} \rightarrow \frac{dg}{dx} = g(x)(1-g(x))$$

$$= -\sum_{i=1}^{N}\left[y^i \frac{1}{g(\mathbf{w}^T\mathbf{x}^i)} - (1-y^i)\frac{1}{1-g(\mathbf{w}^T\mathbf{x}^i)}\right] g(\mathbf{w}^T\mathbf{x}^i)(1-g(\mathbf{w}^T\mathbf{x}^i))\frac{\partial \mathbf{w}^T\mathbf{x}^i}{\partial w_k}$$

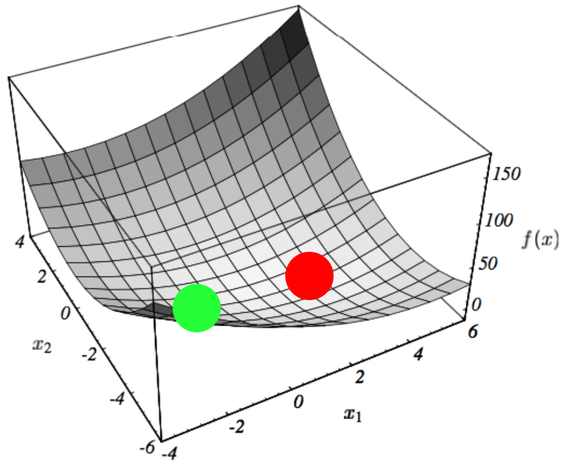
$$= -\sum_{i=1}^{N}\left[y^i(1-g(\mathbf{w}^T\mathbf{x}^i)) - (1-y^i)g(\mathbf{w}^T\mathbf{x}^i)\right] x_k^i$$

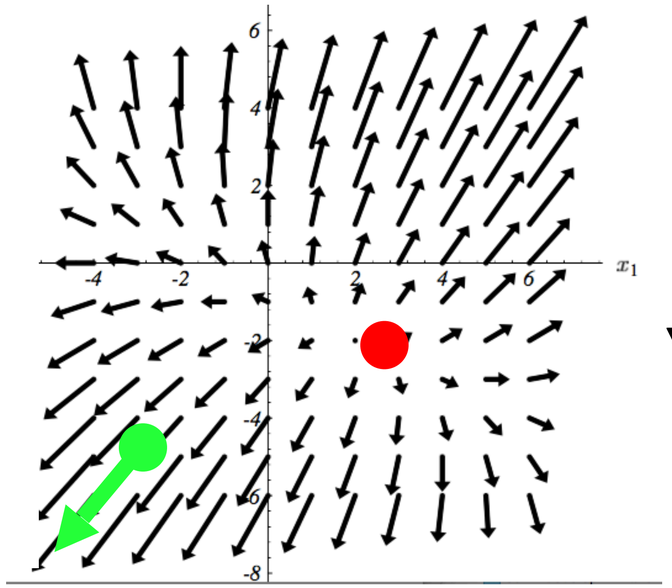
$$= -\sum_{i=1}^{N}\left[y^i - g(\mathbf{w}^T\mathbf{x}^i)\right] \mathbf{x}_k^i$$

$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$

nonlinear system of equations!!

Gradient Descent Minimization

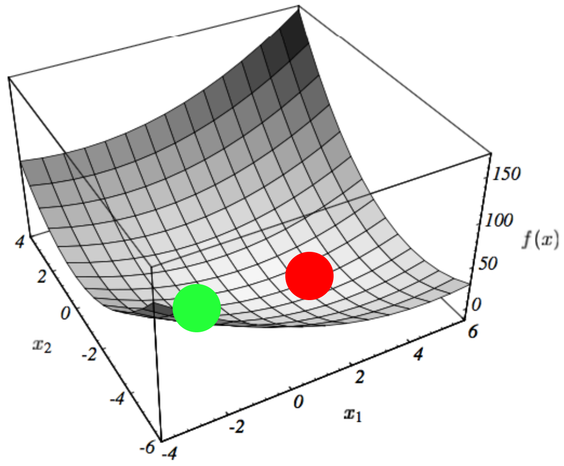


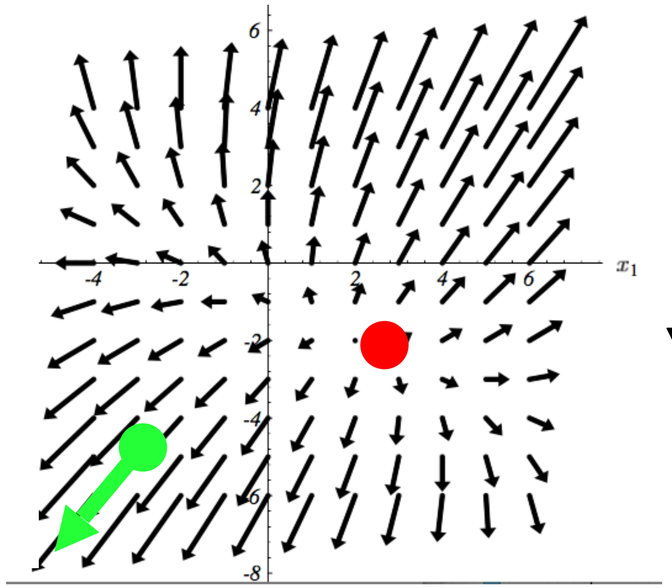


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Gradient Descent Minimization



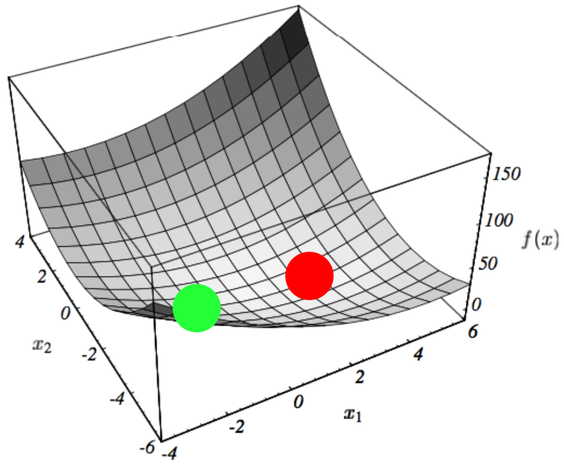


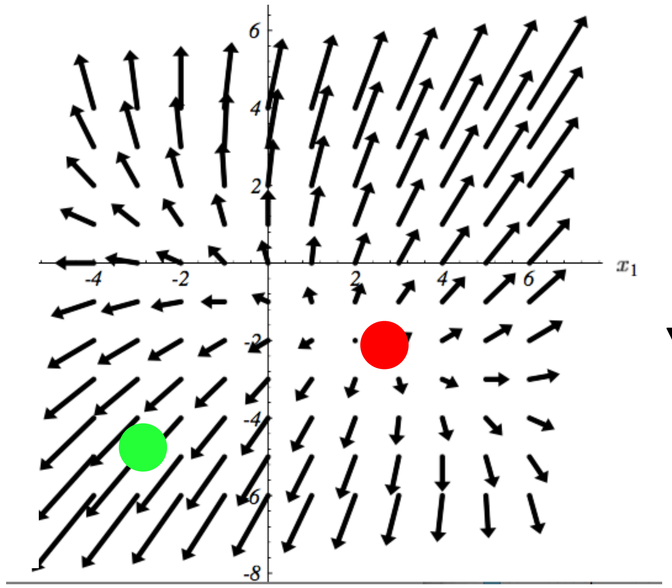
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient Descent Minimization



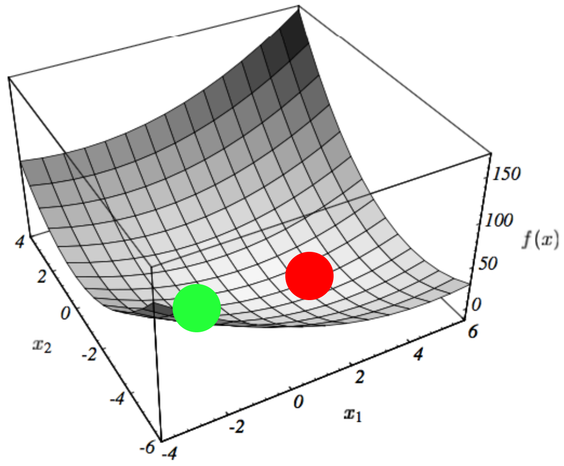


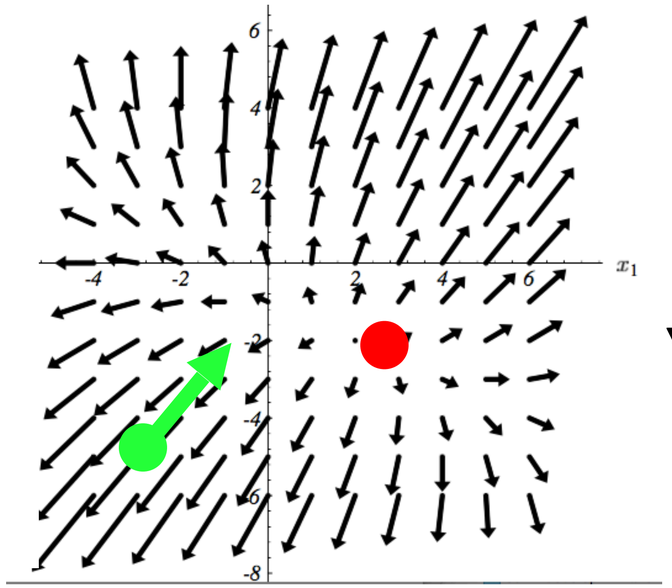
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient Descent Minimization



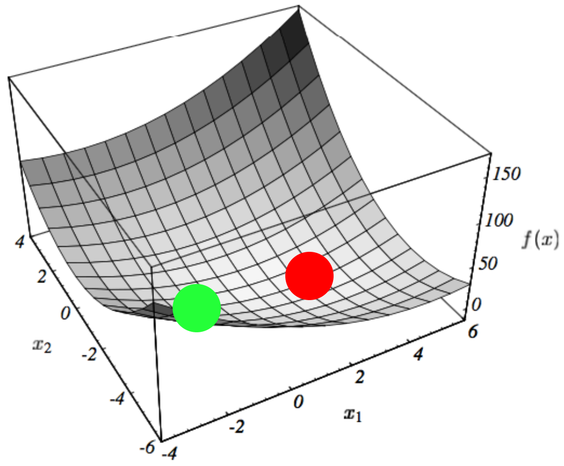


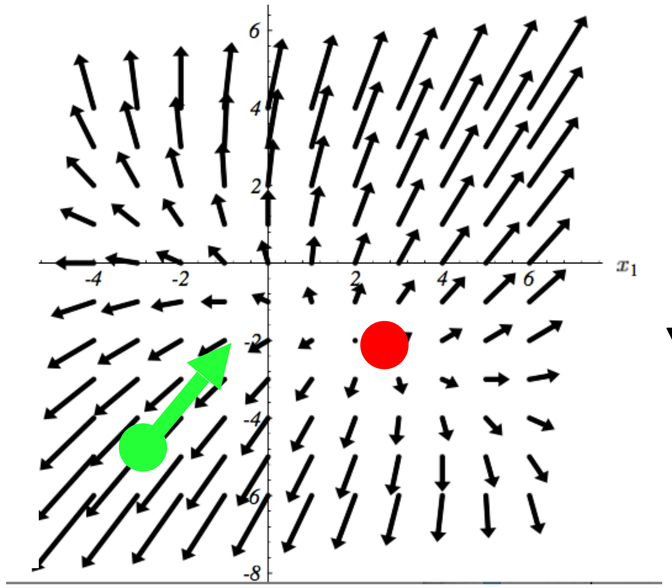
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient Descent Minimization





$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

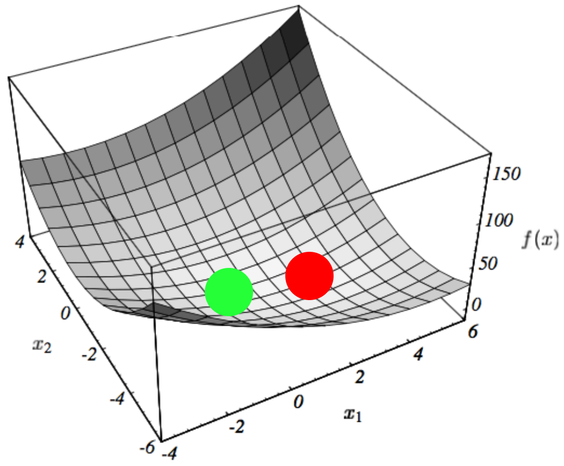
Fact: gradient at any point gives direction of fastest increase

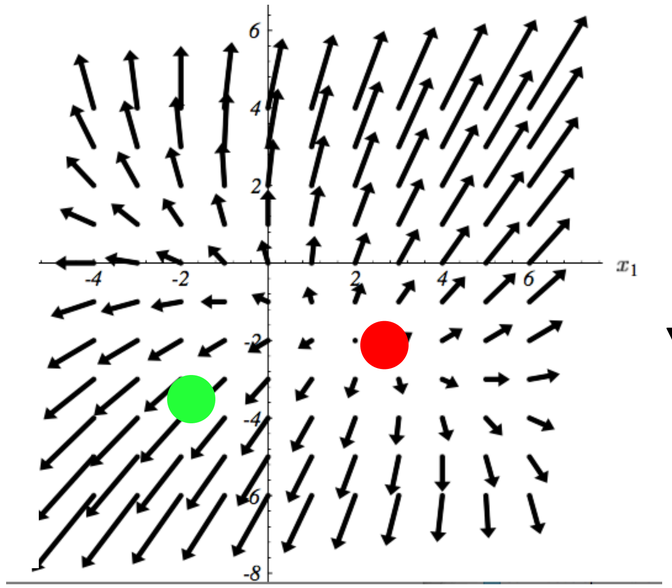
Idea: start at a point and move in the direction opposite to the gradient

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ **i=0**

Gradient Descent Minimization

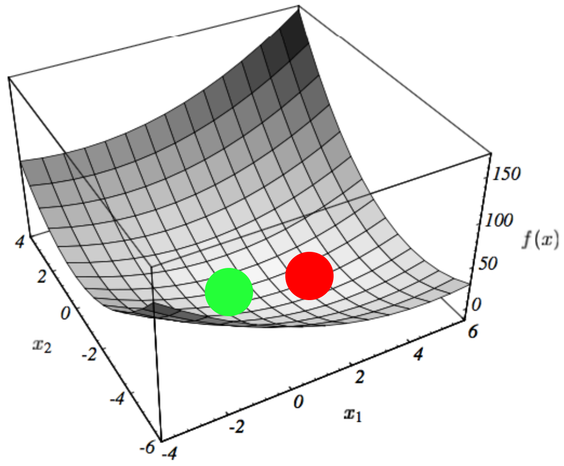


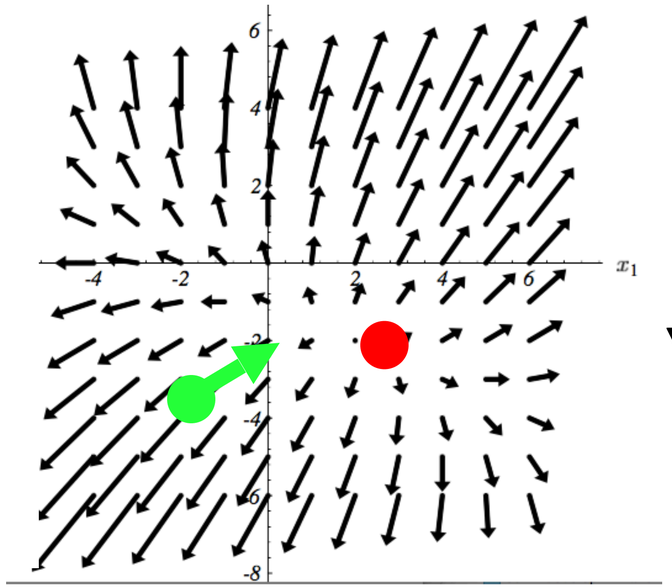


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=1**

Gradient Descent Minimization

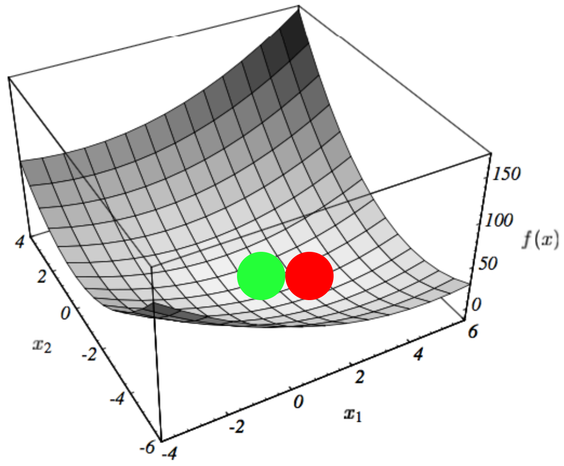


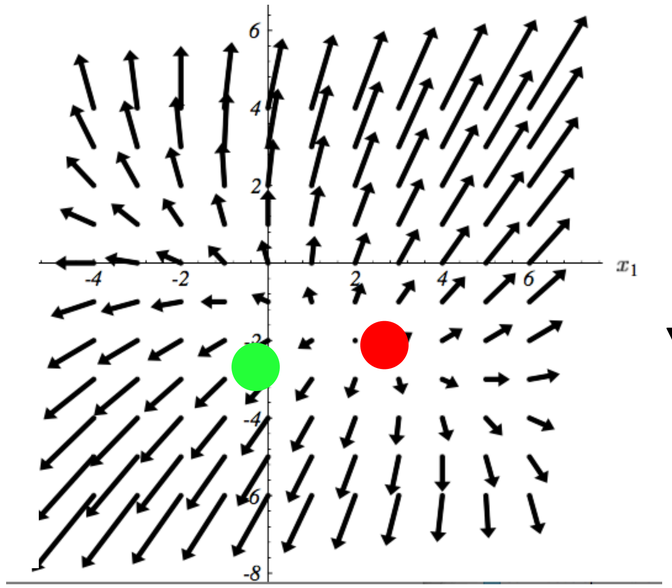


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=1**

Gradient Descent Minimization

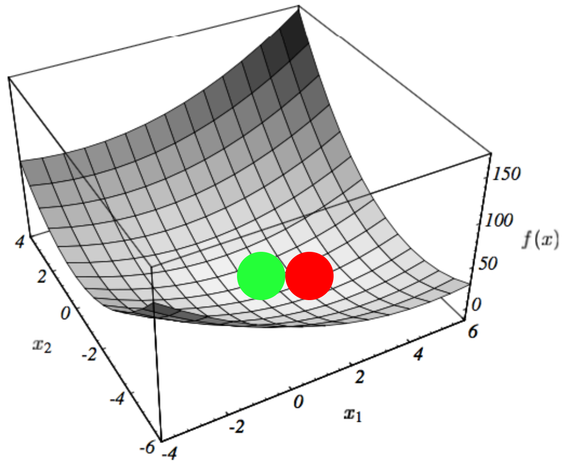


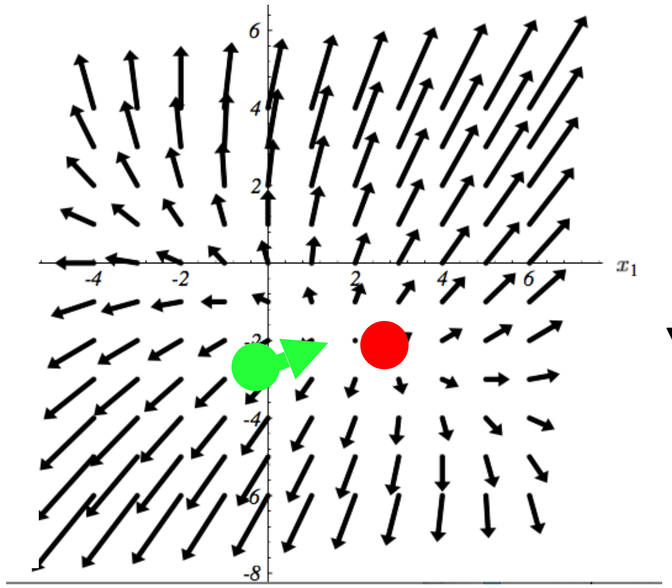


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=2**

Gradient Descent Minimization

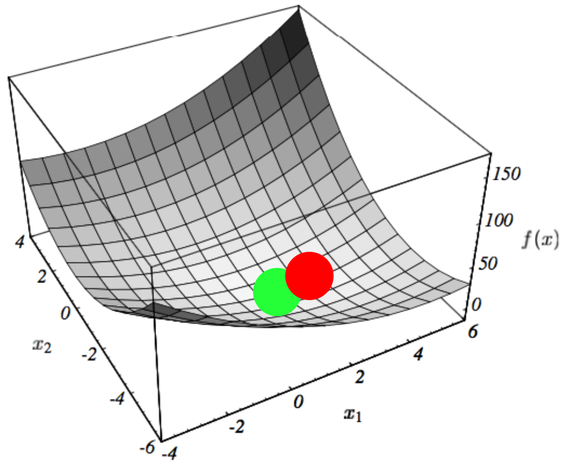


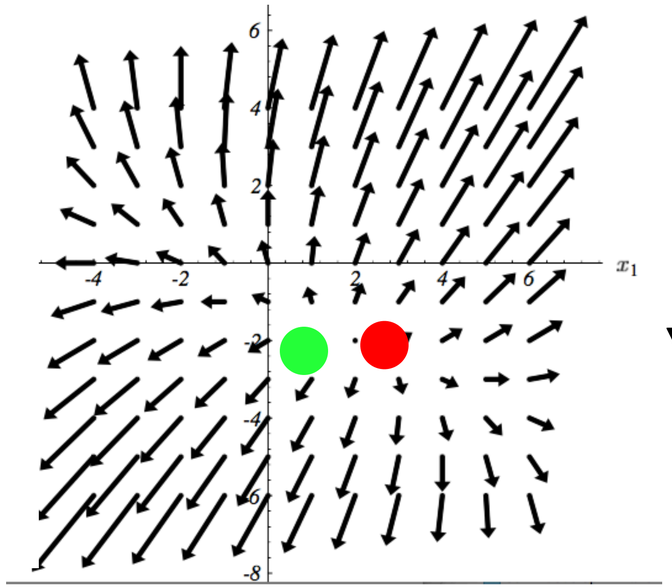


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=2**

Gradient Descent Minimization





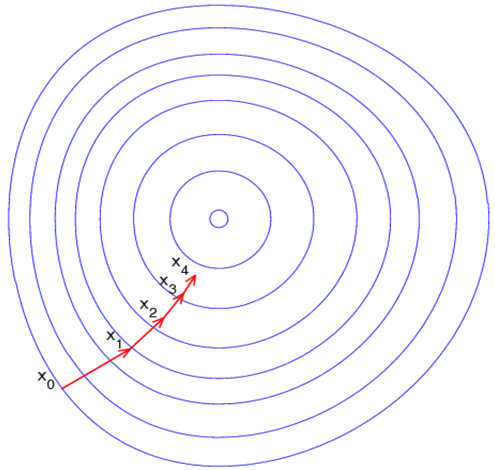
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=3**

Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

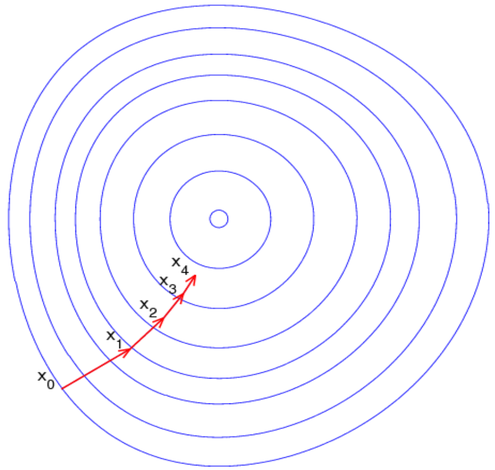


Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

We can always make it converge for a convex function.

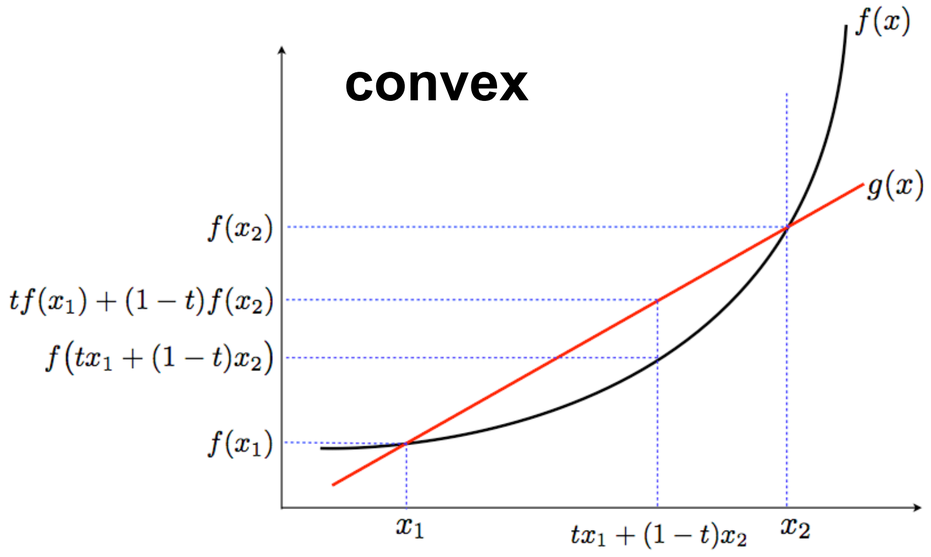


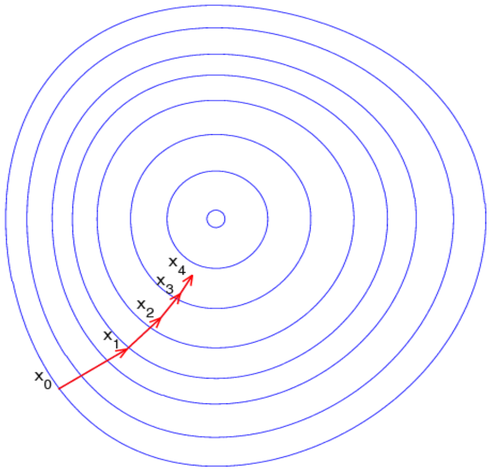
Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

We can always make it converge for a convex function.



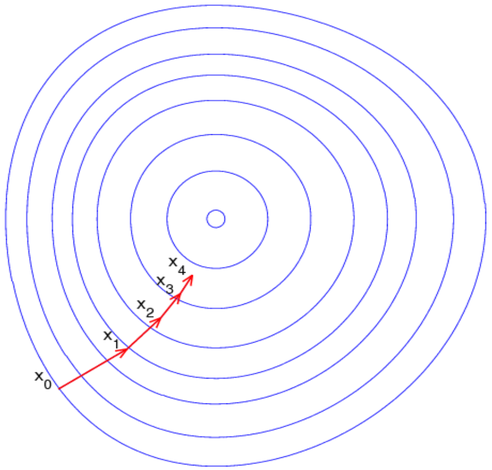


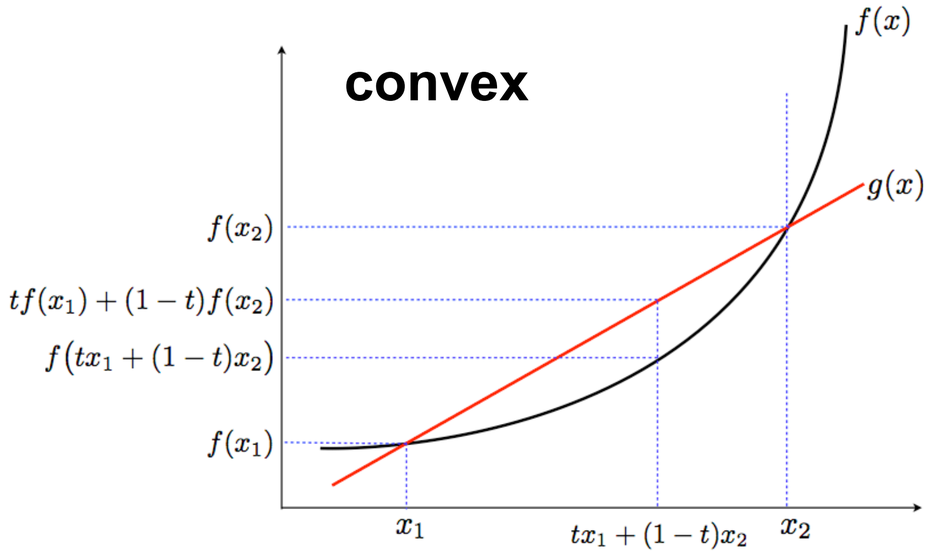
Gradient Descent Minimization

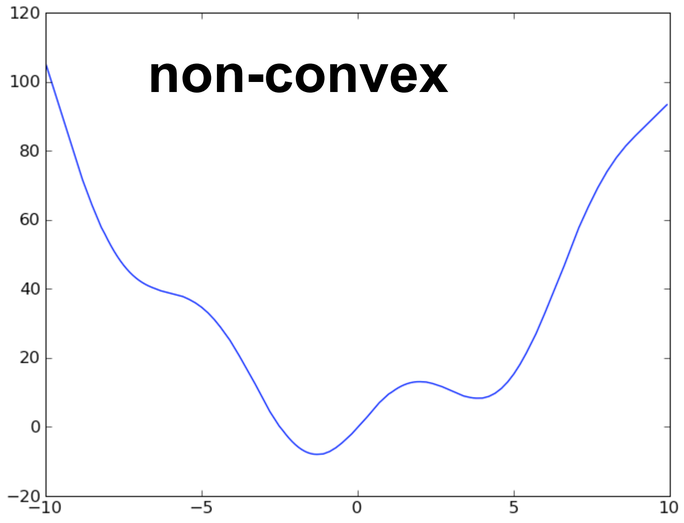
Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

We can always make it converge for a convex function.







XOR Problem

XOR Problem

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

XOR Problem

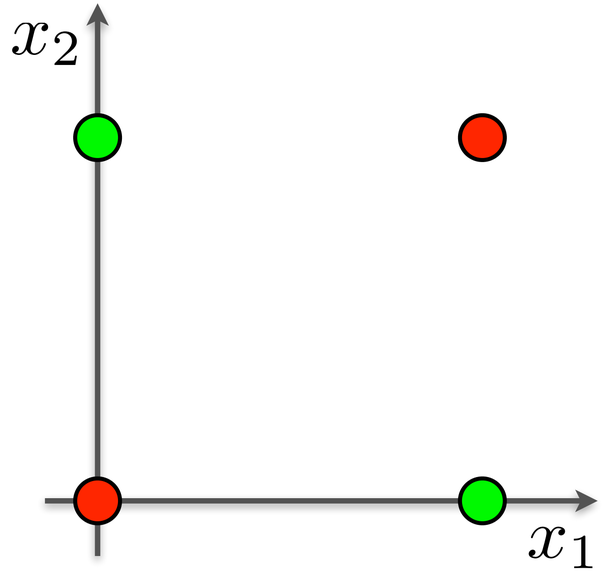
$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

XOR Problem

$$y = f(x_1, x_2)$$

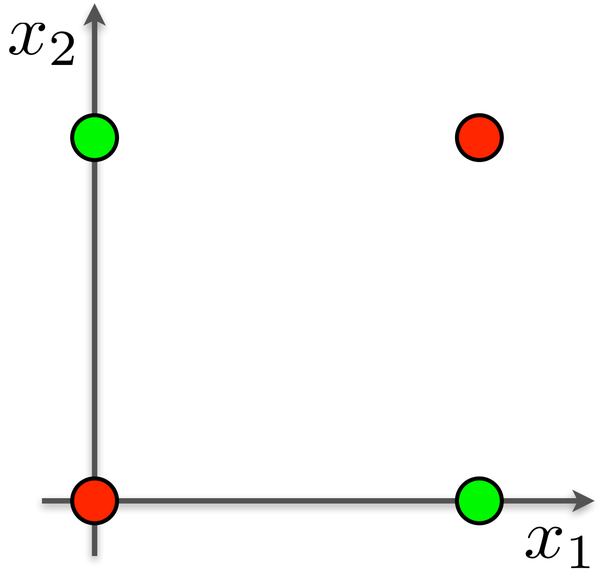
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



XOR Problem

$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

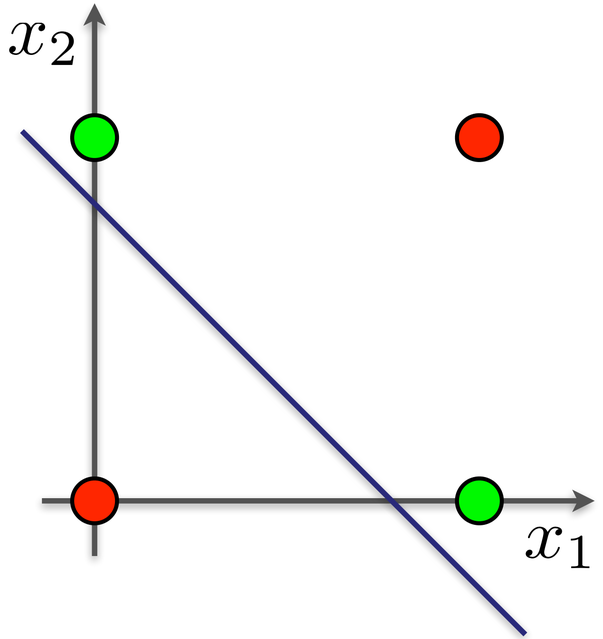


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

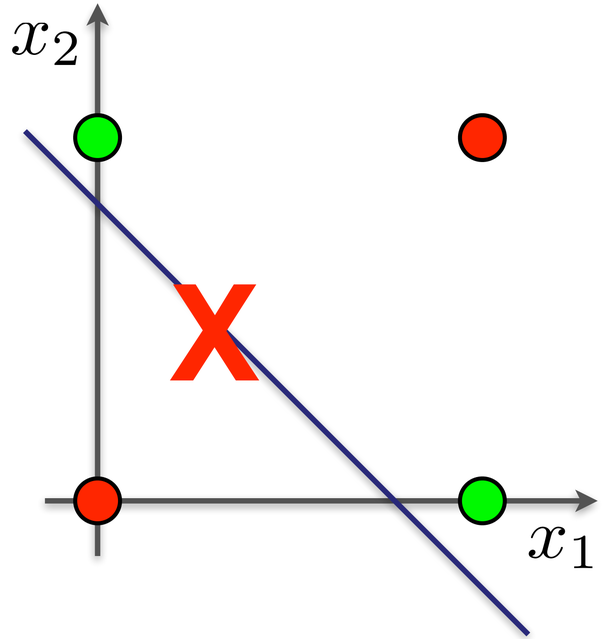


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

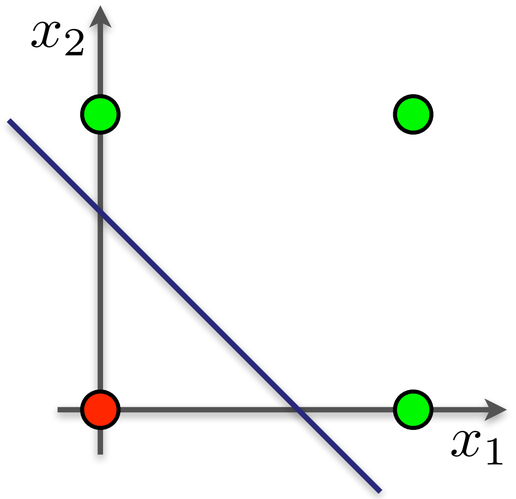


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

XOR Problem

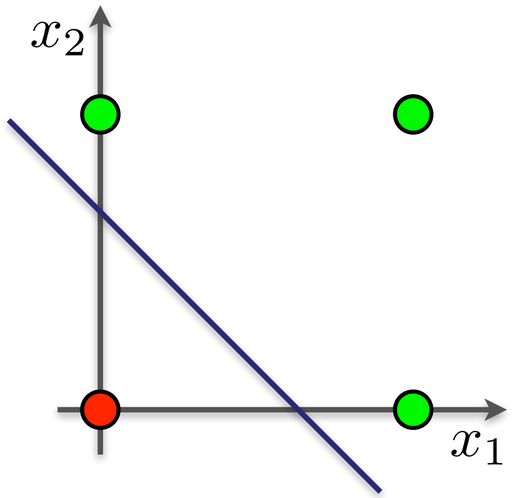
x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1

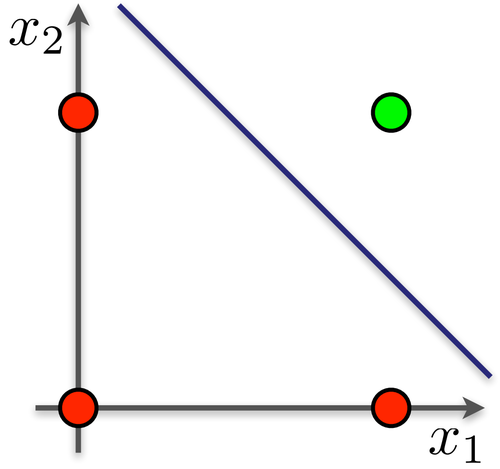


XOR Problem

x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1

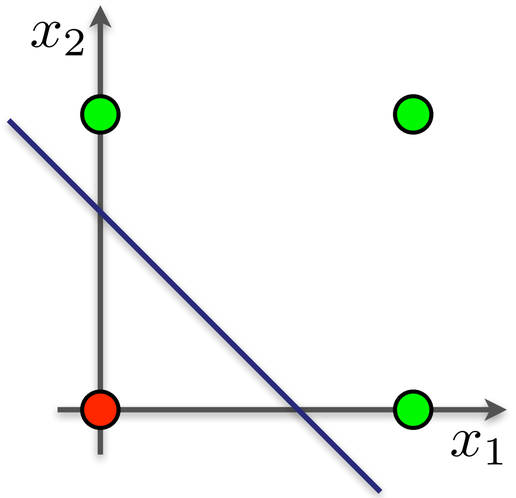
x_1	x_2	z_2
0	0	0
0	1	0
1	0	0
1	1	1



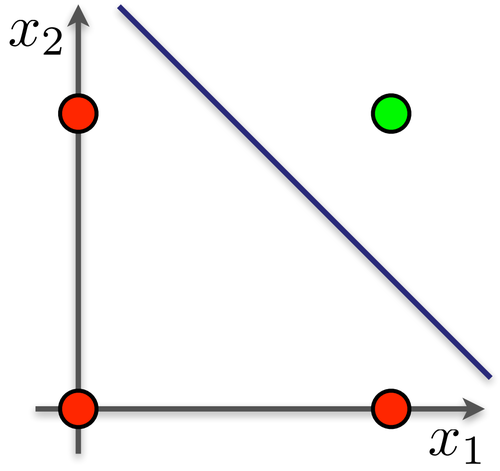


XOR Problem

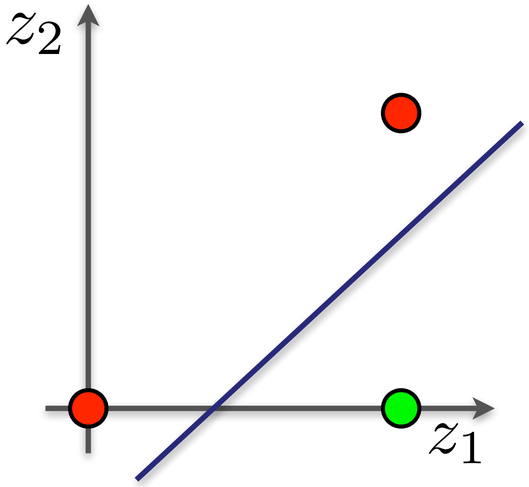
x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1



x_1	x_2	z_2
0	0	0
0	1	0
1	0	0
1	1	1



z_1	z_2	y
0	0	0
1	0	1
1	0	1
1	1	0



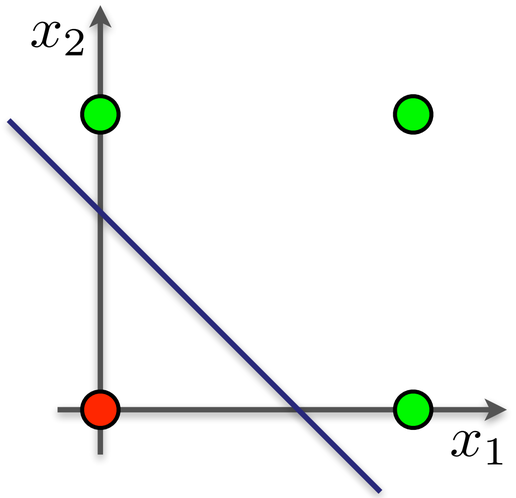
XOR Problem

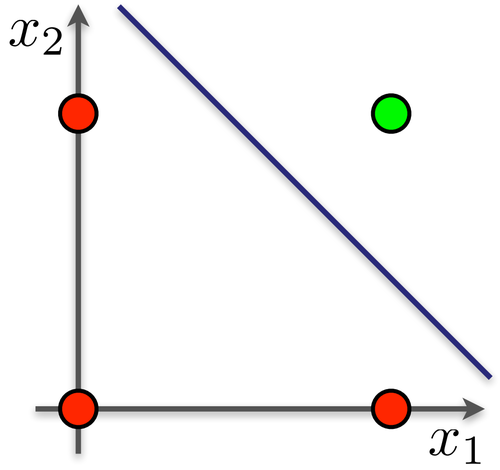
$$y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2))$$

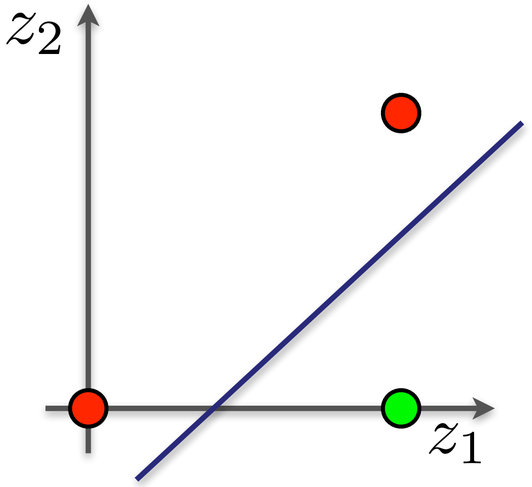
x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	z_2
0	0	0
0	1	0
1	0	0
1	1	1

z_1	z_2	y
0	0	0
1	0	1
1	0	1
1	1	0

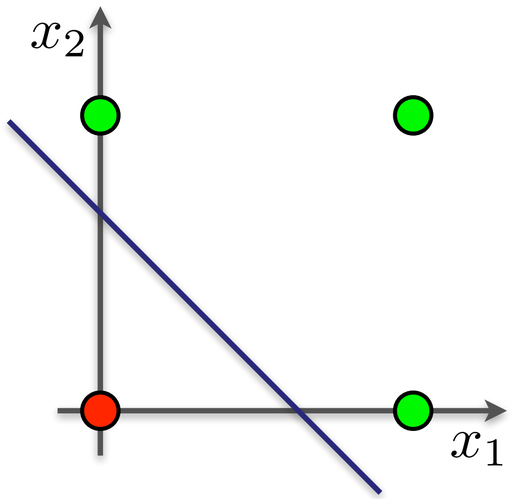


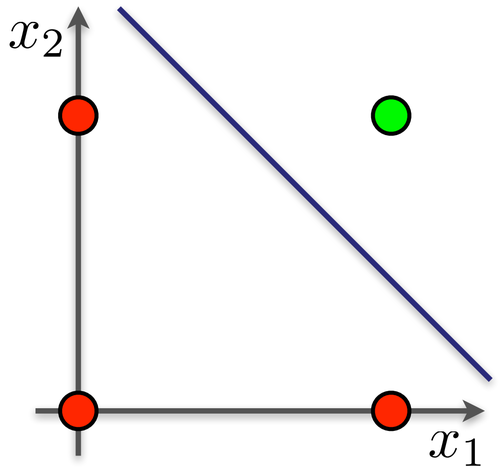


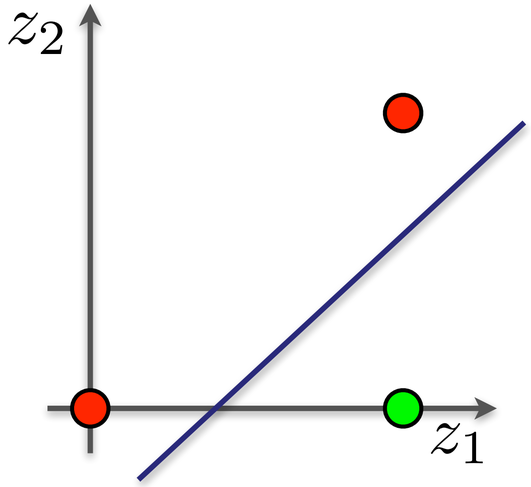


XOR Problem

$$y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2))$$

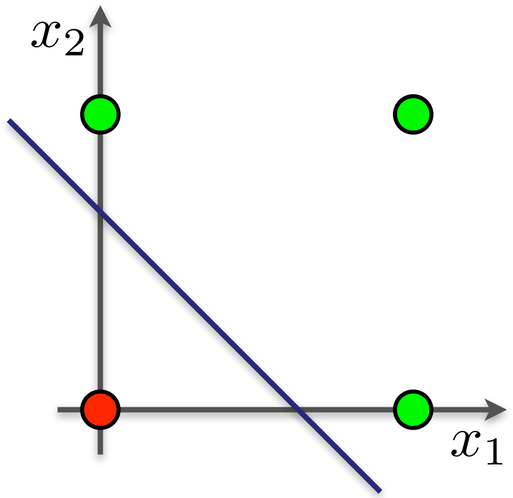


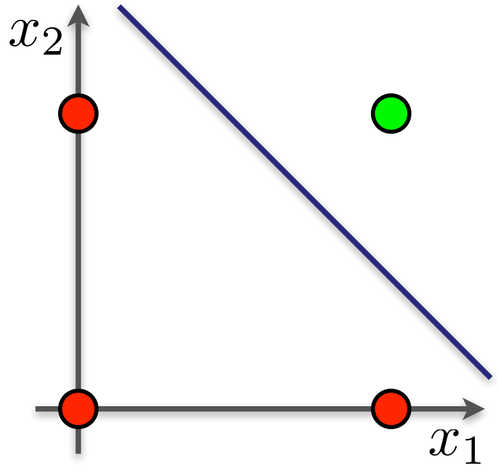


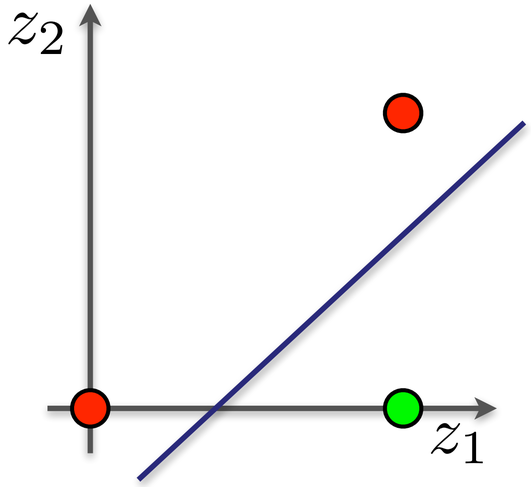


XOR Problem

$$y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2))$$
$$= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right)$$

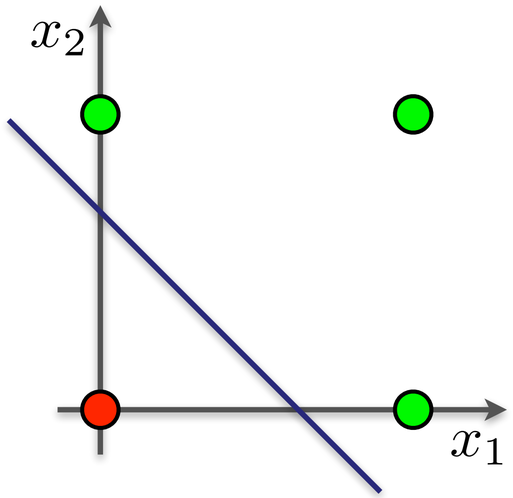


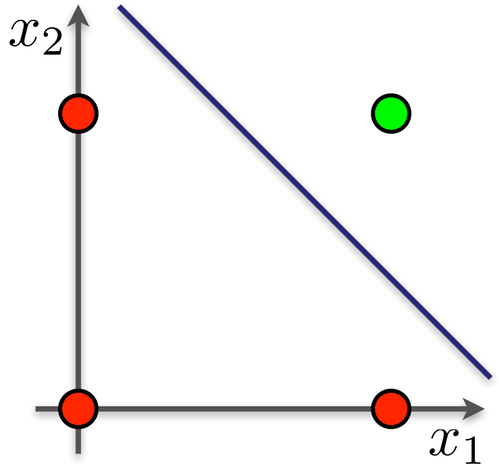


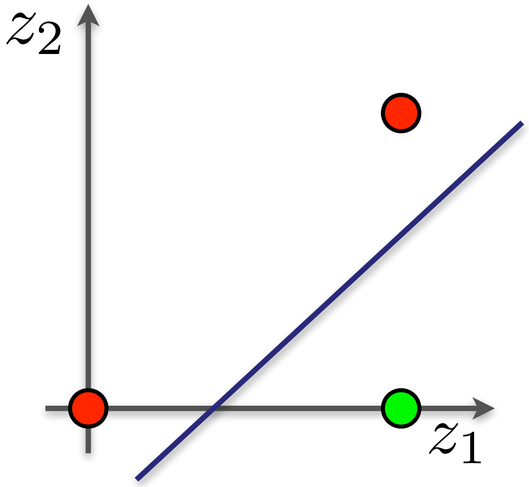


XOR Problem

$$
\begin{aligned}
y = f(z_1, z_2) &= f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\
&= \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2))
\end{aligned}
$$

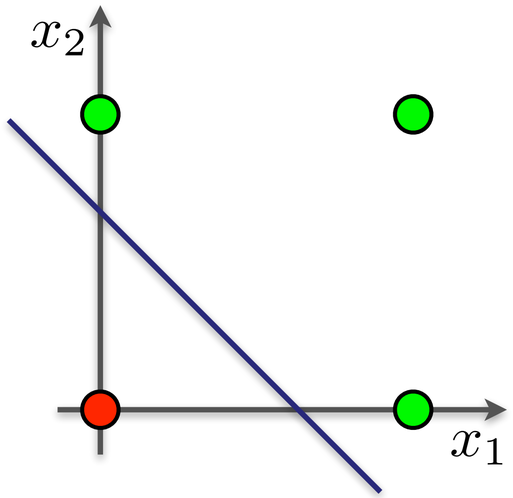


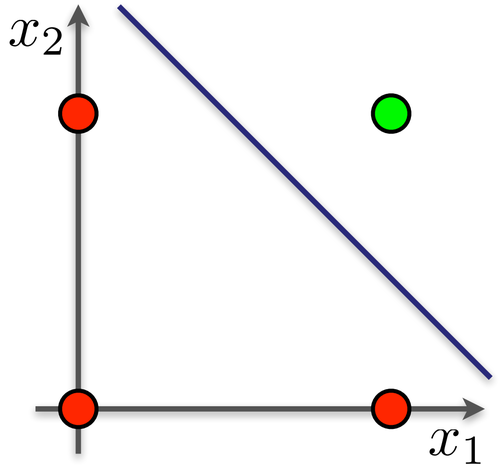


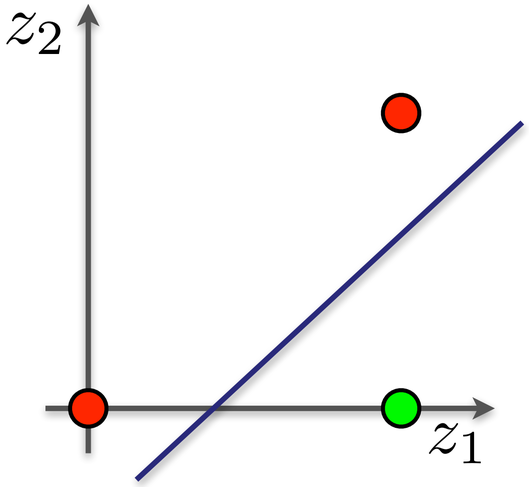


XOR Problem

$$
\begin{aligned}
y = f(z_1, z_2) &= f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\
&= \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2)))
\end{aligned}
$$

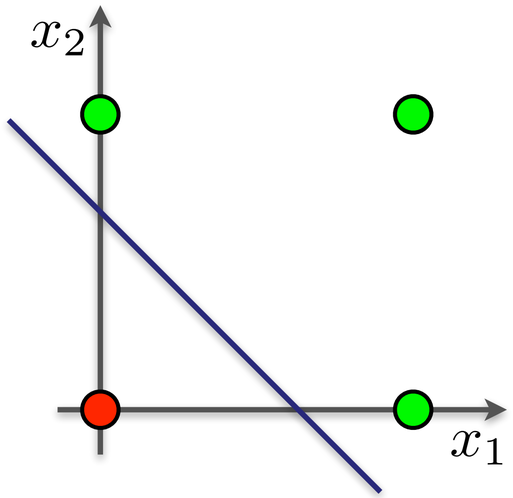


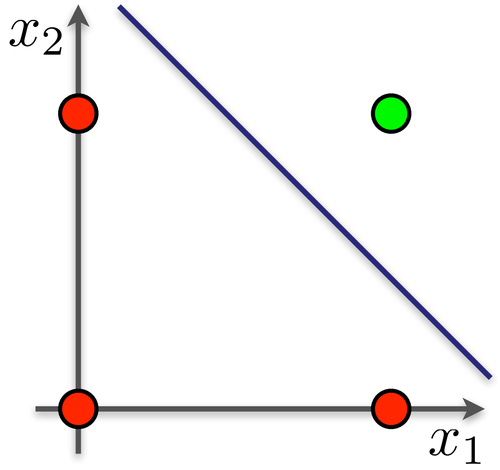


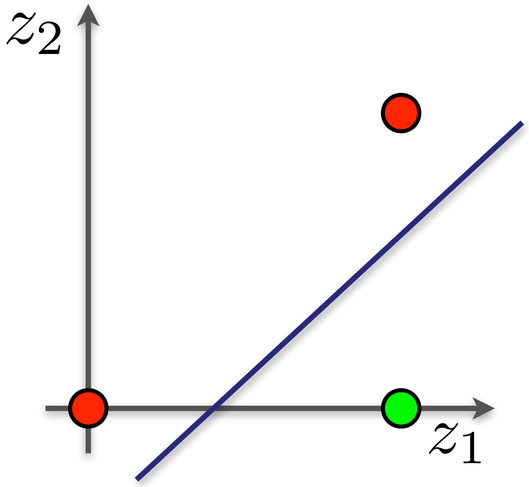


XOR Problem

$$
\begin{aligned}
y = f(z_1, z_2) &= f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\
&= \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2)))
\end{aligned}
$$





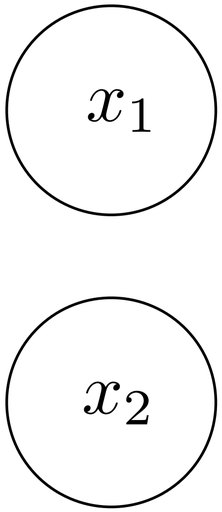


XOR Problem

$$
\begin{aligned}
y = f(z_1, z_2) &= f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\
&= \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2)))
\end{aligned}
$$

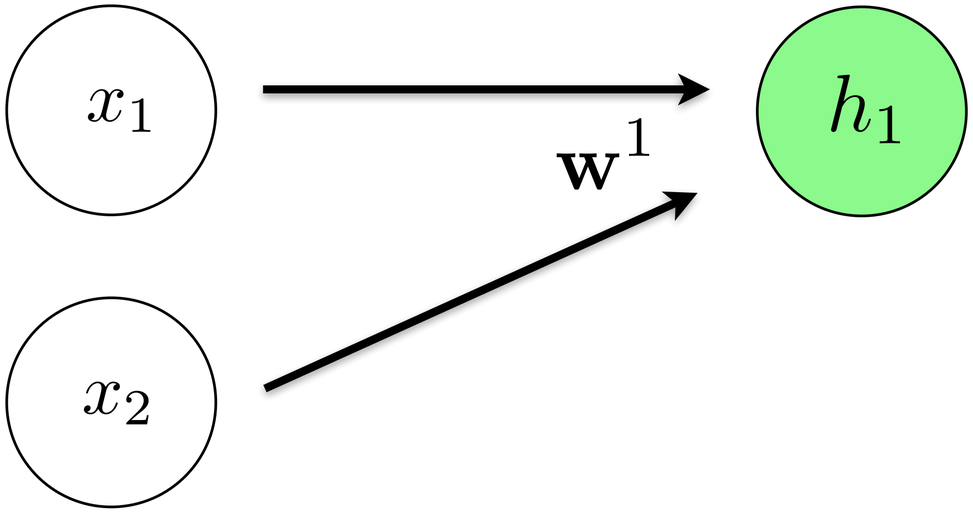
XOR Problem

$$
\begin{aligned}
y = f(z_1, z_2) &= f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\
&= \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2)))
\end{aligned}
$$



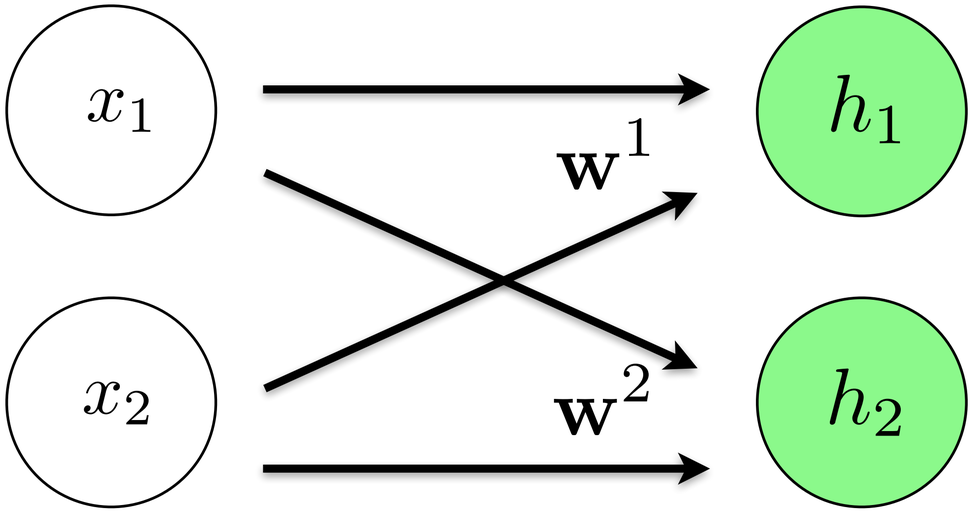
XOR Problem

$$
\begin{aligned}
y = f(z_1, z_2) &= f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\
&= \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2)))
\end{aligned}
$$



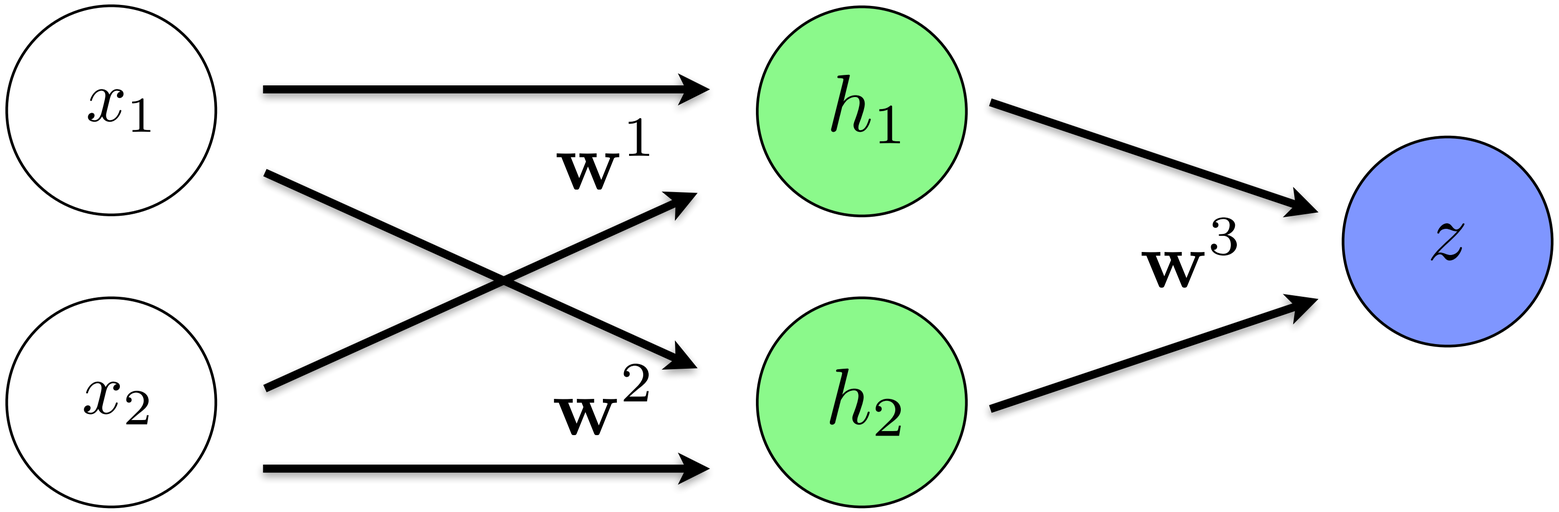
XOR Problem

$$
\begin{aligned}
y = f(z_1, z_2) &= f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\
&= \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2)))
\end{aligned}
$$

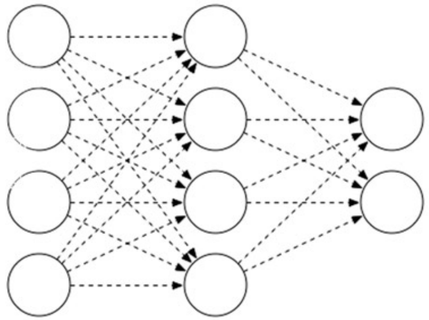


XOR Problem

$$
\begin{aligned}
y = f(z_1, z_2) &= f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\
&= \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2)))
\end{aligned}
$$



Course Information (slides/code/comments)



http://geometry.cs.ucl.ac.uk/dl4g/