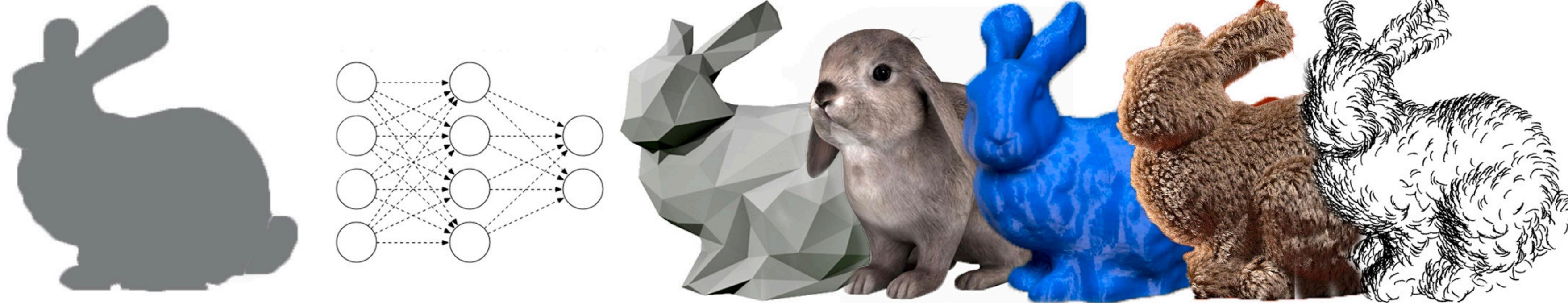


COMP0169: Machine Learning for Visual Computing

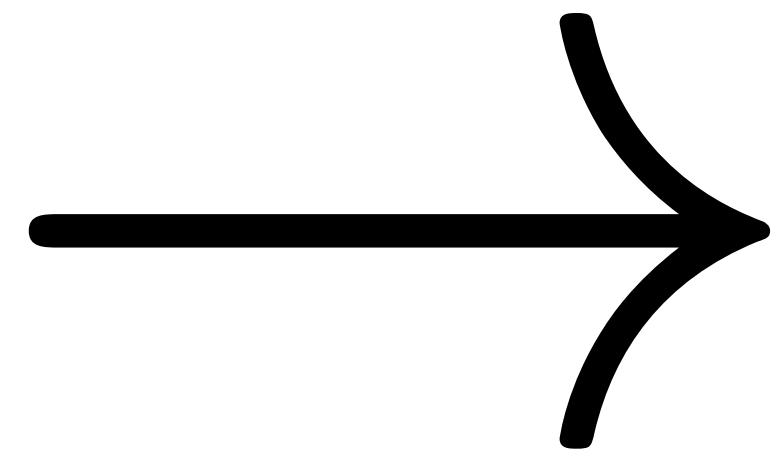
kNN (k Nearest Neighbors) And Regression



Lectures will be Recorded

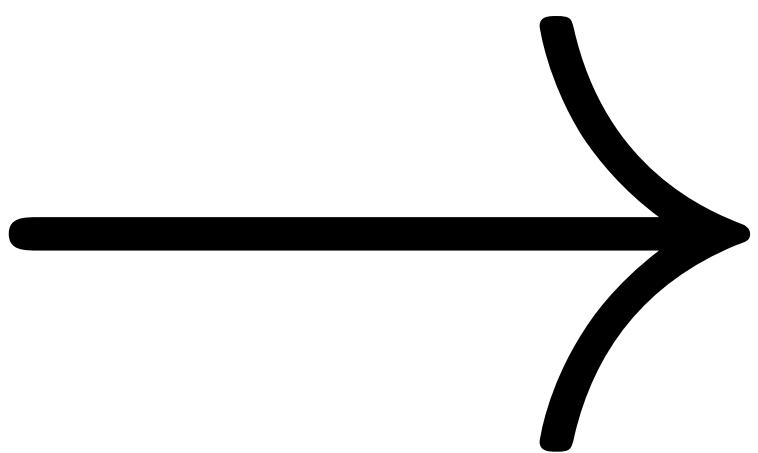
Image as a Point

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167



$$\mathbf{p} \in \mathbb{R}^{3WH}$$

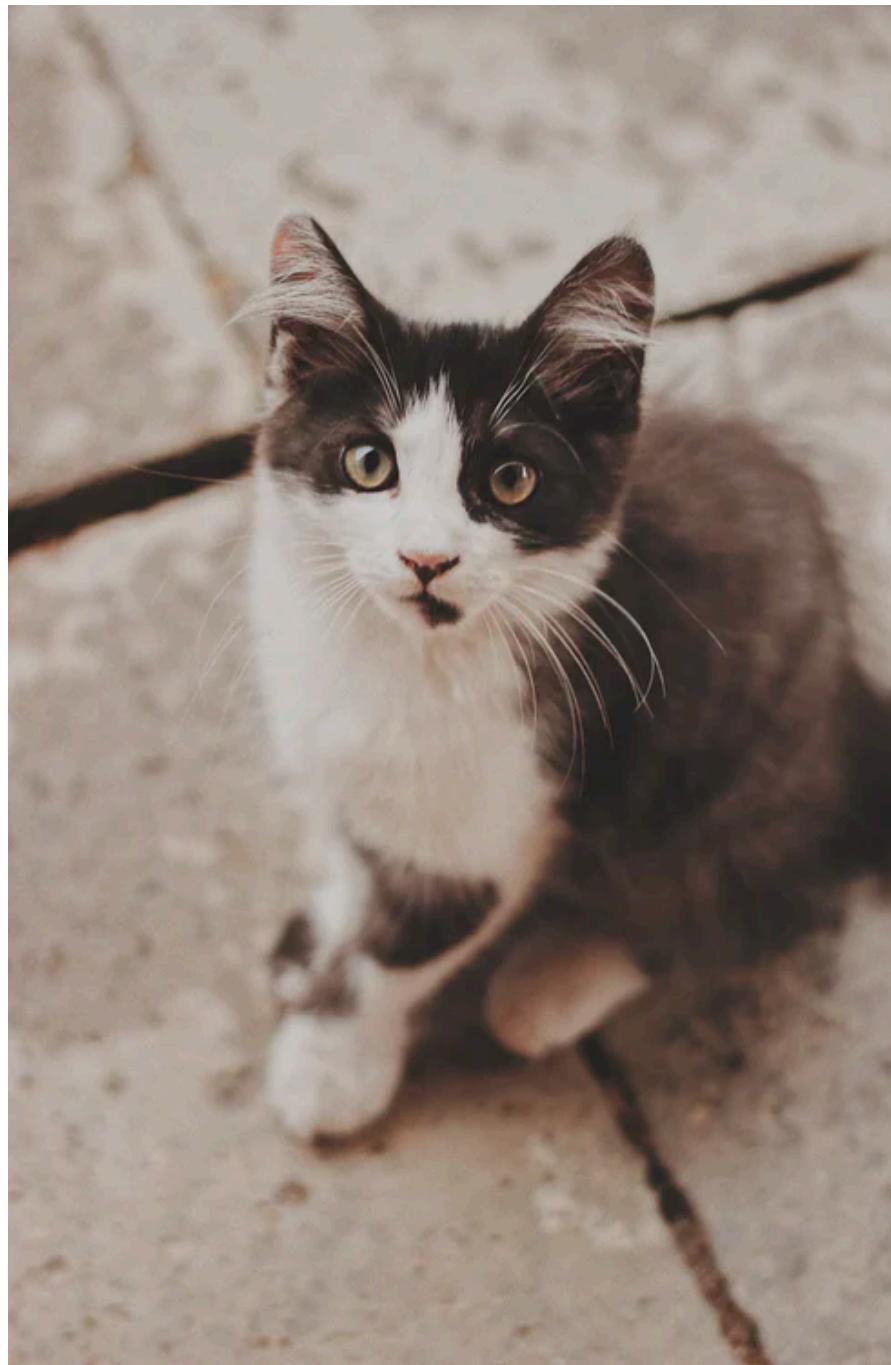
Image as a Point


$$\mathbf{p} \in \mathbb{R}^{3WH}$$

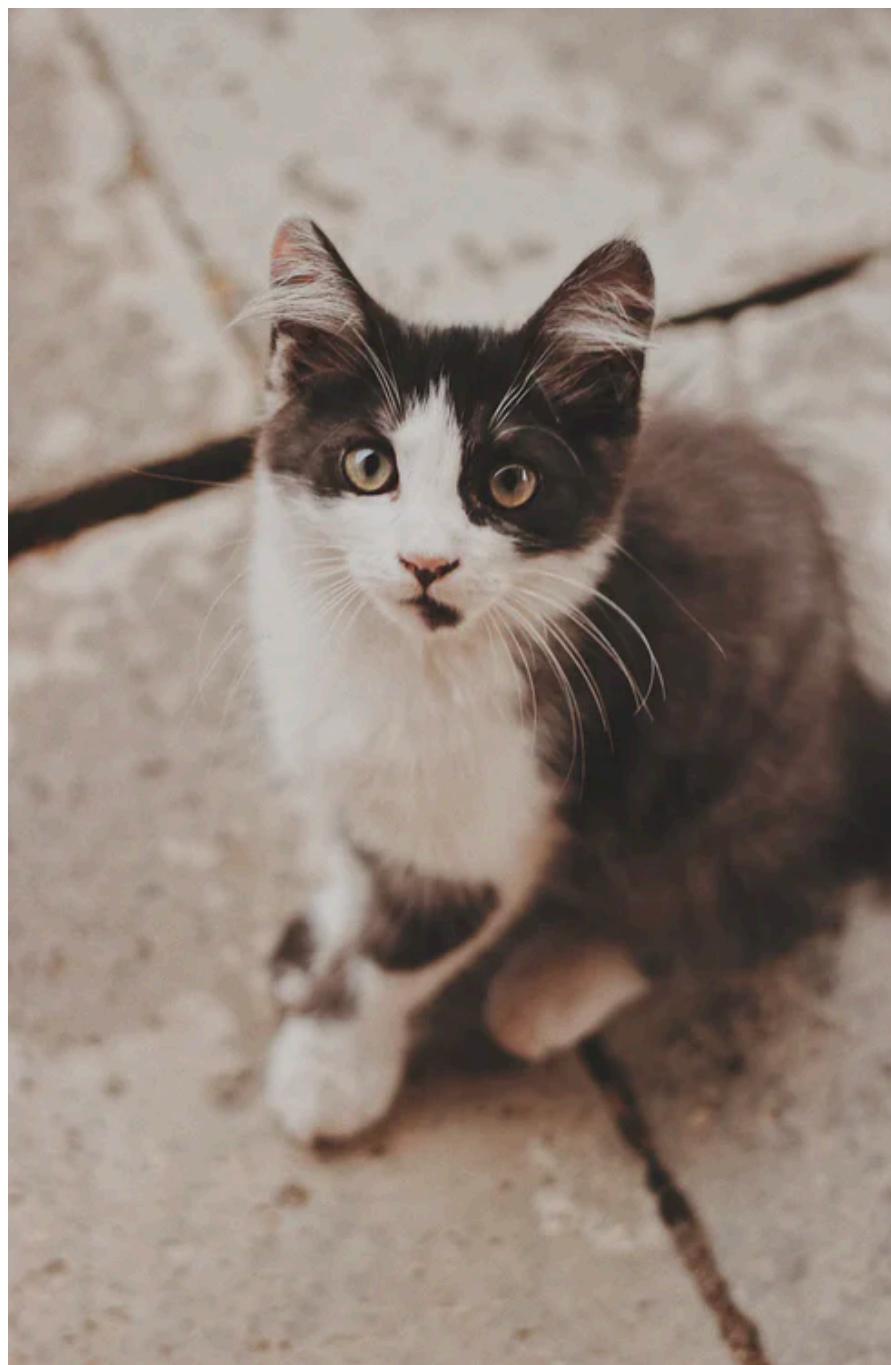
MidPoint of Images?



MidPoint of Images?



Distance between Images?

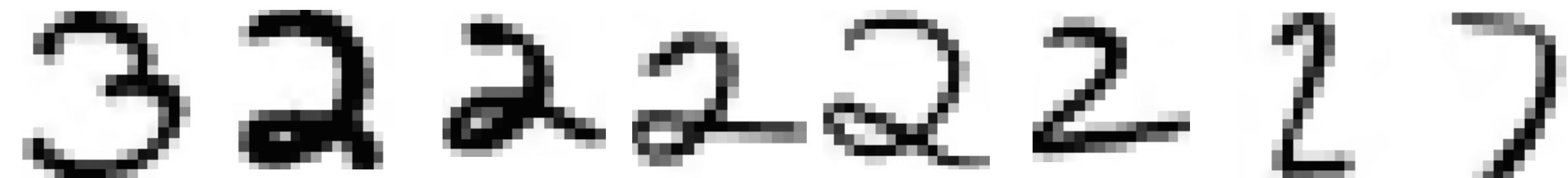


Machine Learning Variants

- **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

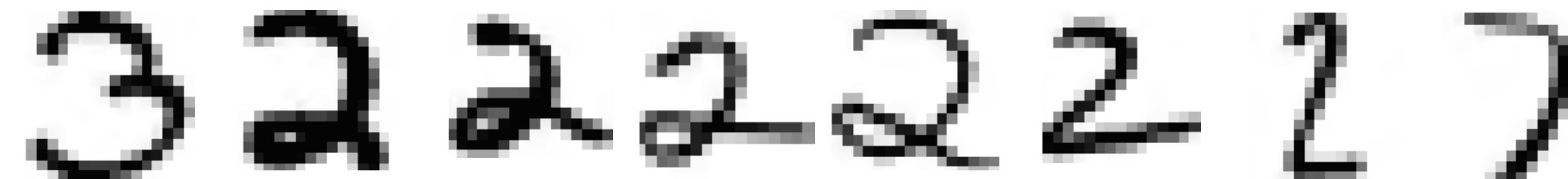
Classification Examples

- Digit Recognition



Classification Examples

- Digit Recognition

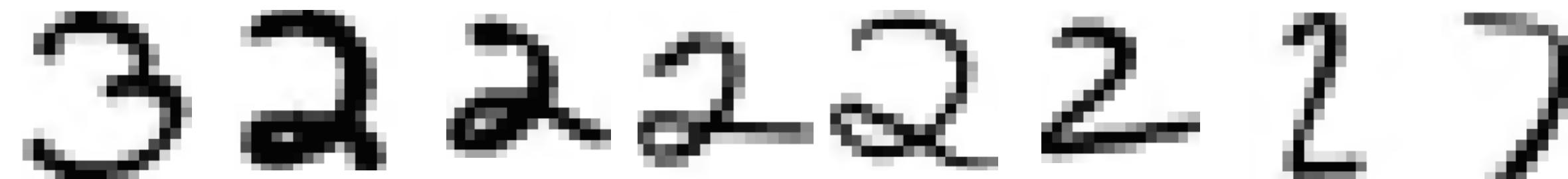


- Spam Detection

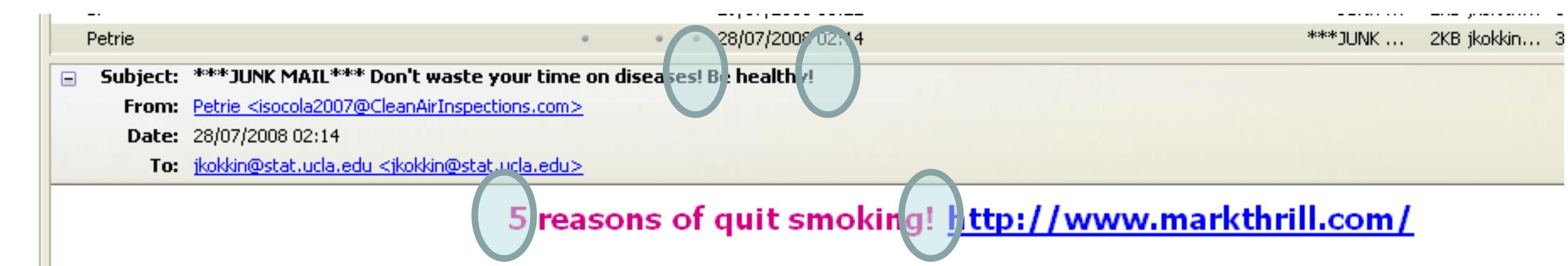


Classification Examples

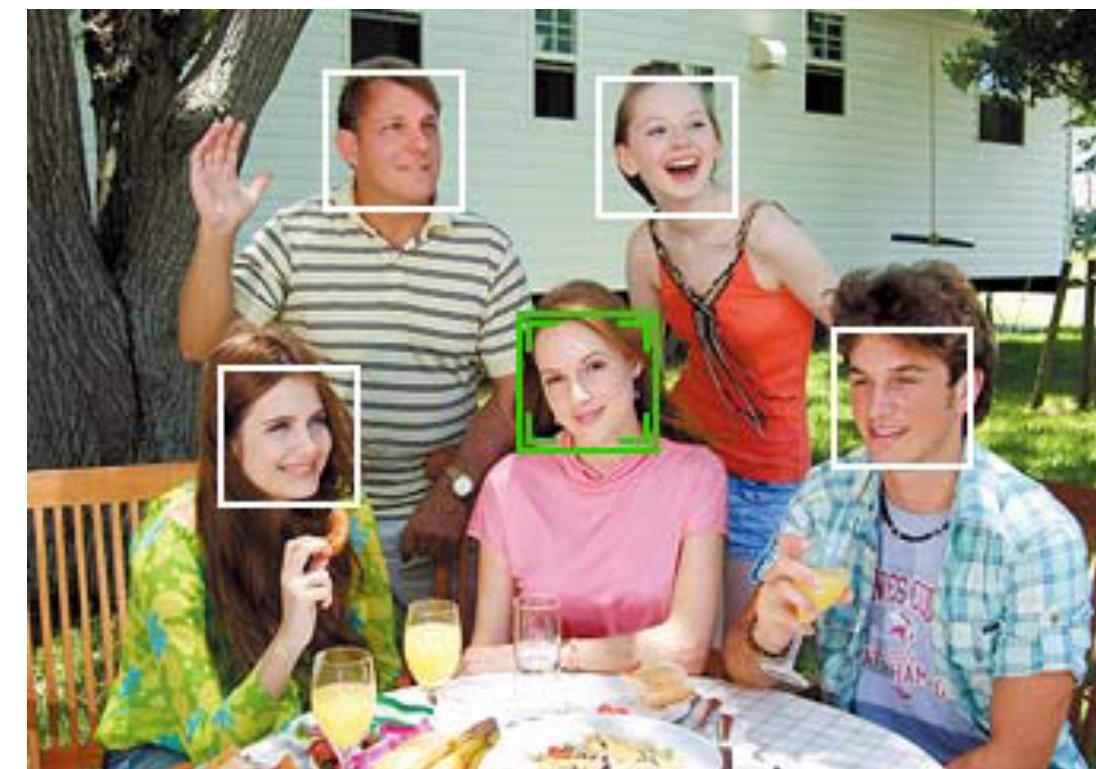
- **Digit Recognition**



- **Spam Detection**



- **Face detection**



‘Faceness’ Function: Classifier

background

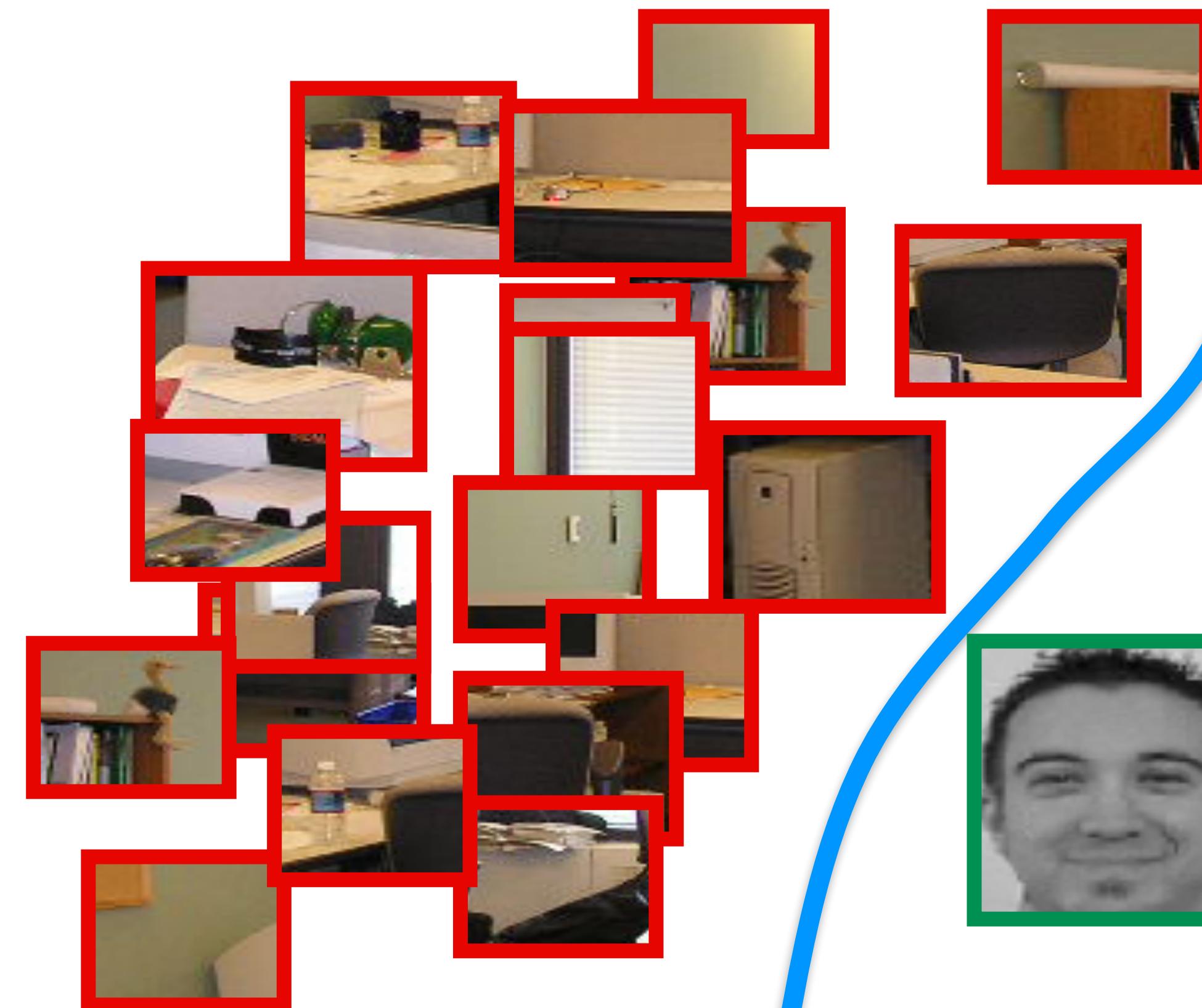


face



‘Faceness’ Function: Classifier

background



decision boundary



face

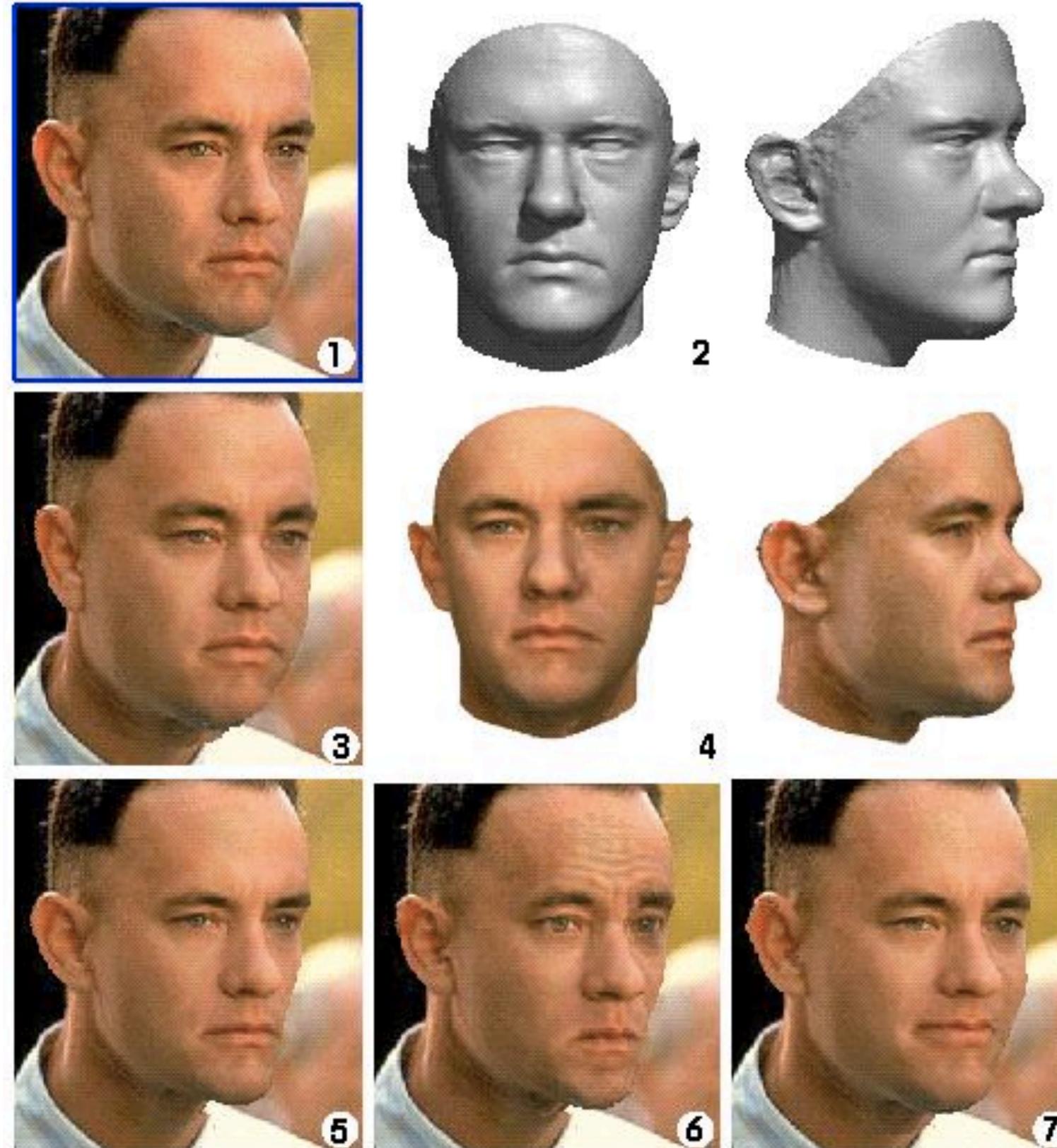


9

Machine Learning Variants

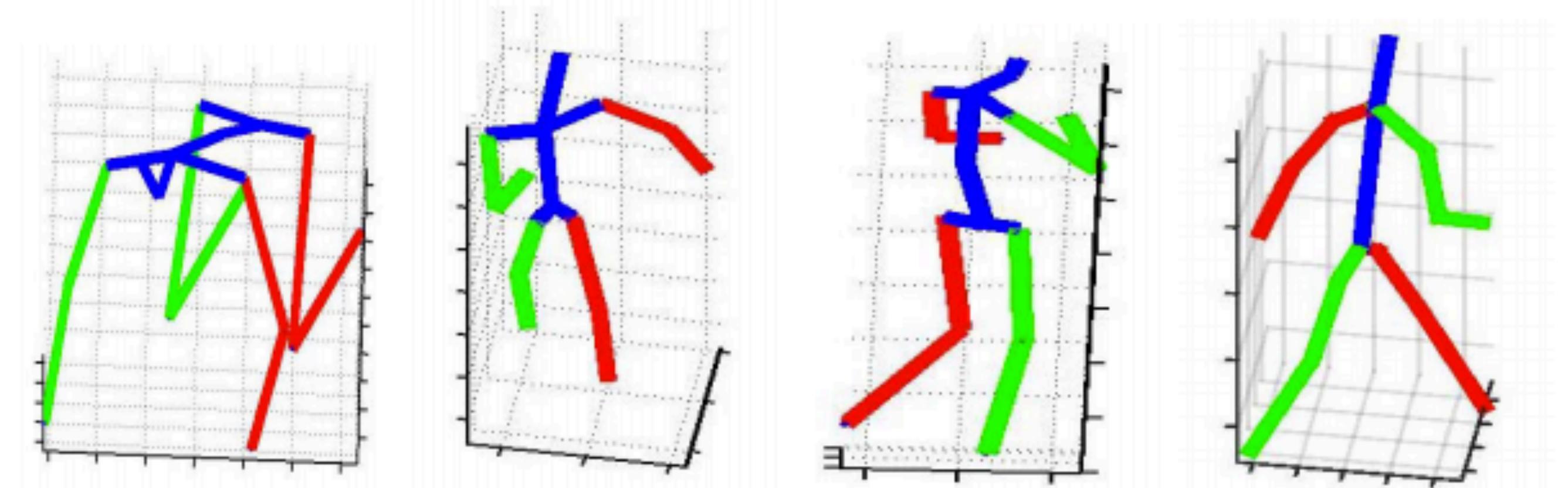
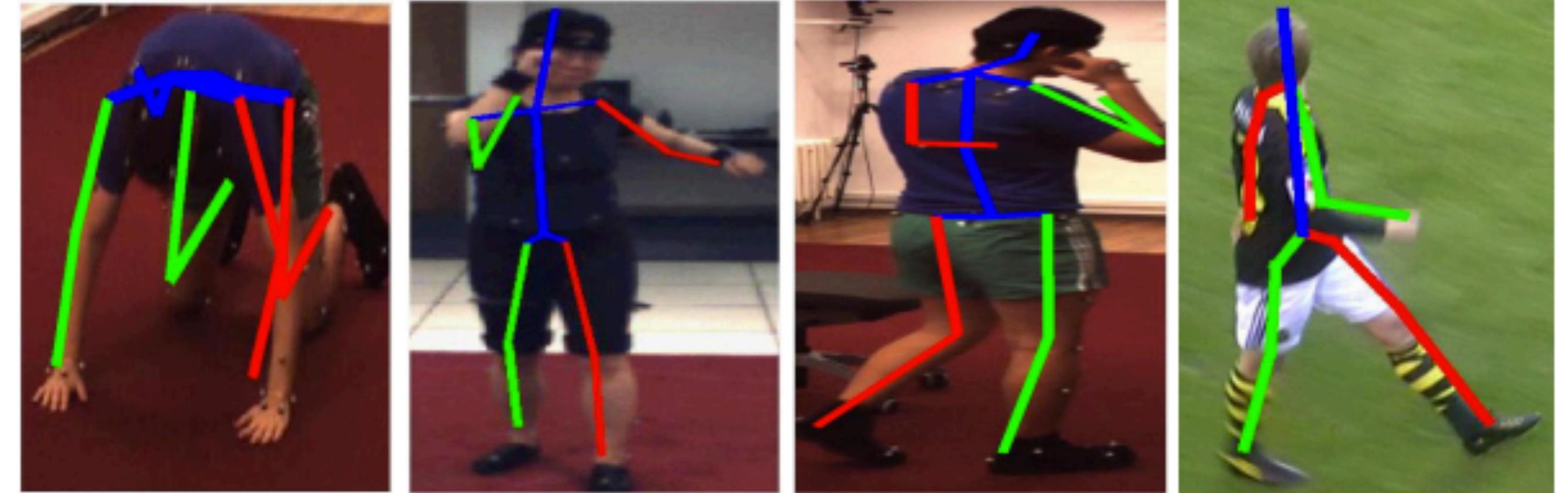
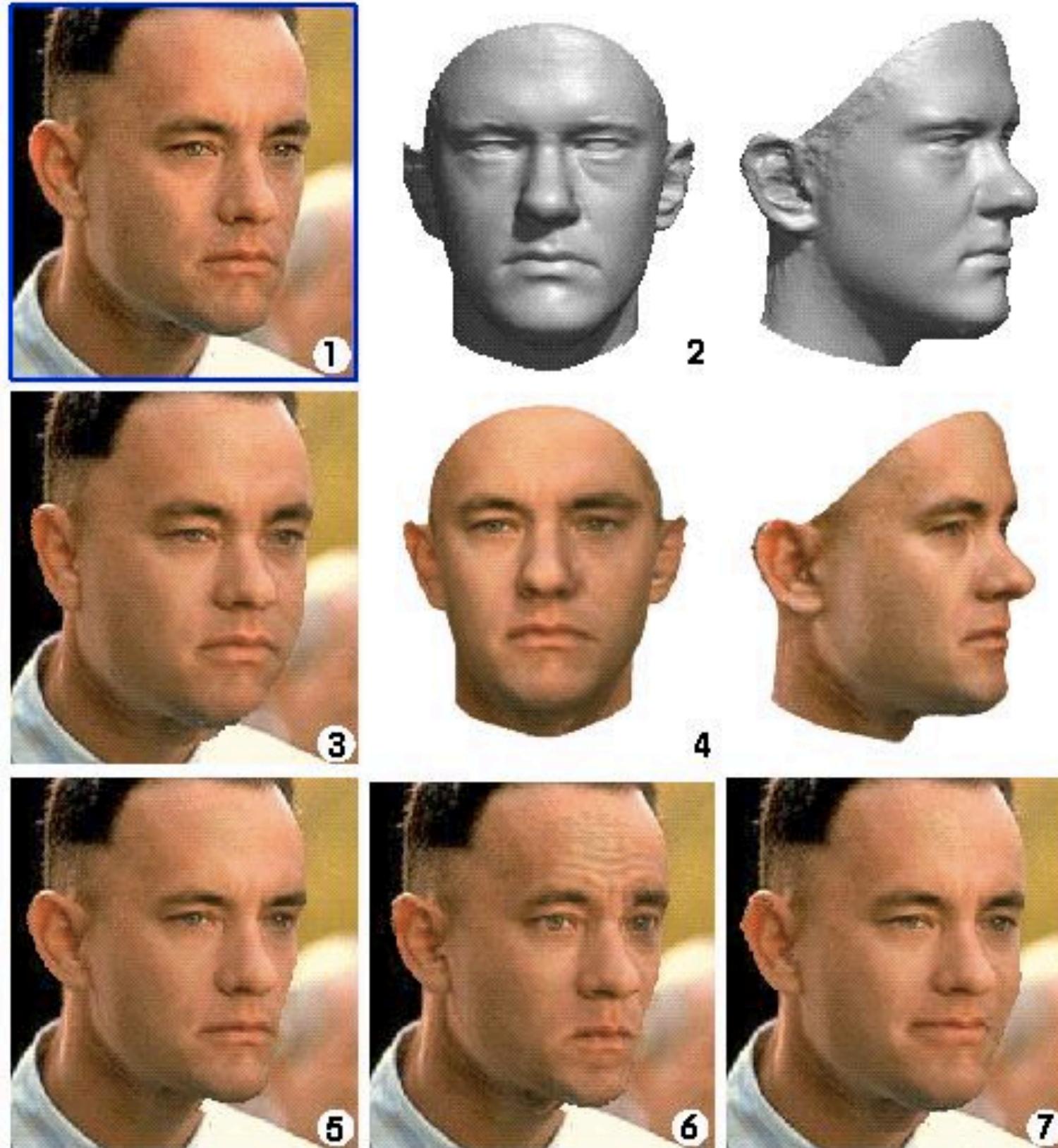
- **Supervised**
 - **Classification**
 - **Regression**
 - **Data consolidation**
- **Unsupervised**
 - **Clustering**
 - **Dimensionality Reduction**
- **Weakly supervised/semi-supervised**
Some data supervised, some unsupervised
- **Reinforcement learning**
Supervision: sparse reward for a sequence of decisions

Human Face/Pose Estimation



[Blanz and Vetter, Siggraph, 1999]

Human Face/Pose Estimation

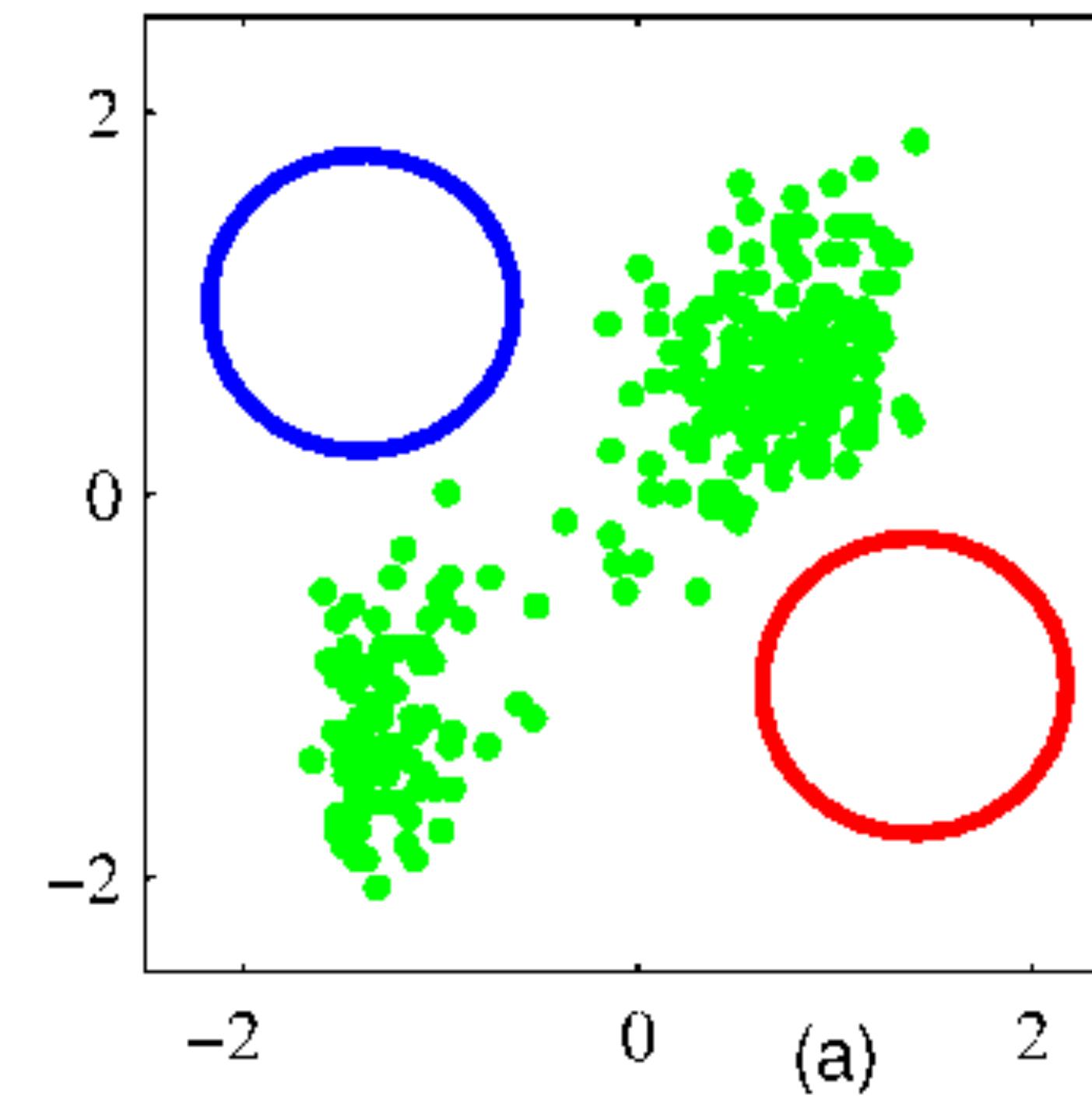


[Blanz and Vetter, Siggraph, 1999]

Machine Learning Variants

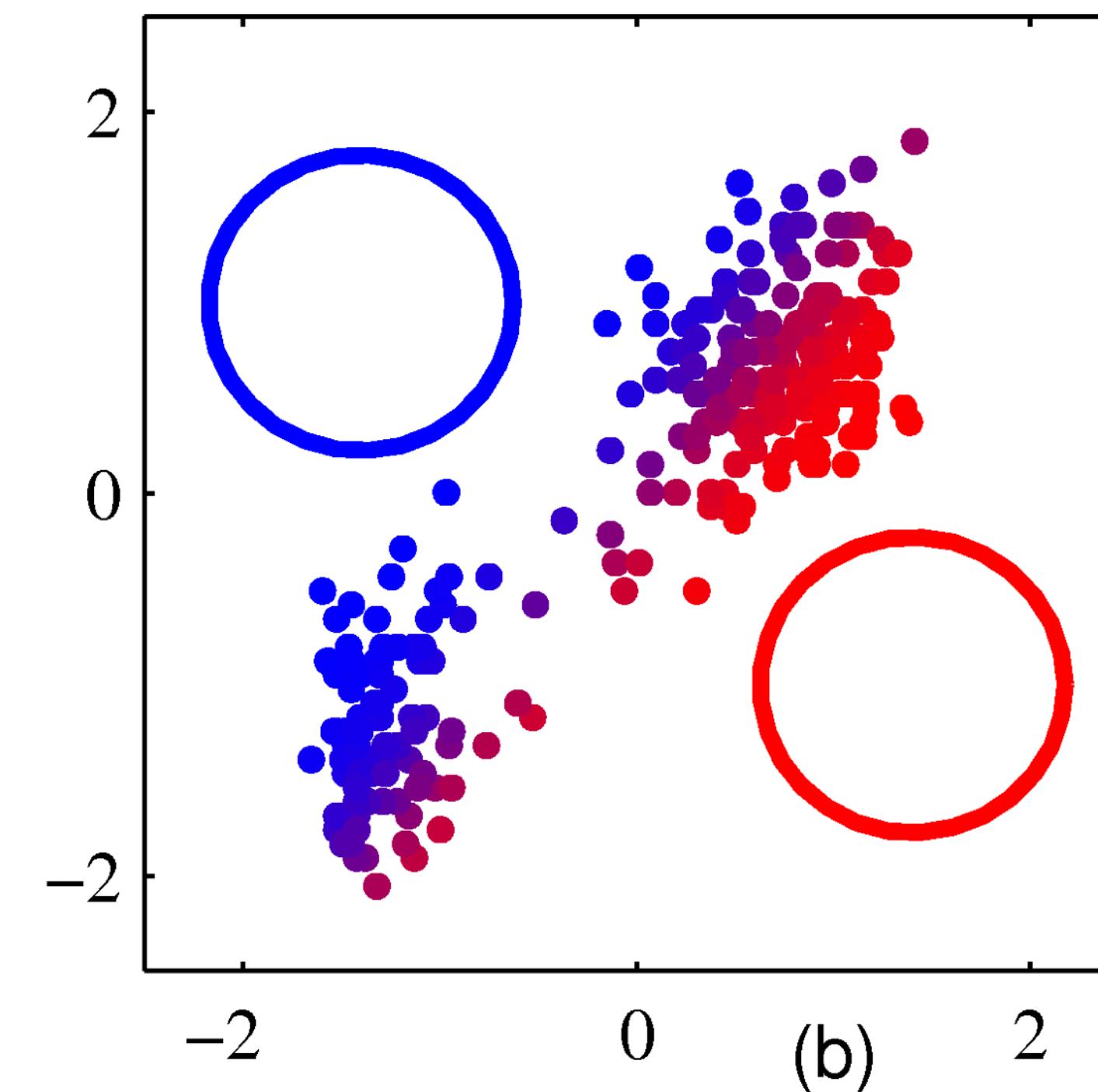
- **Supervised**
 - **Classification**
 - **Regression**
 - **Data consolidation**
- **Unsupervised**
 - **Clustering**
 - **Dimensionality Reduction**
- **Weakly supervised/semi-supervised**
Some data supervised, some unsupervised
- **Reinforcement learning**
Supervision: sparse reward for a sequence of decisions

Clustering: Color Points according to X

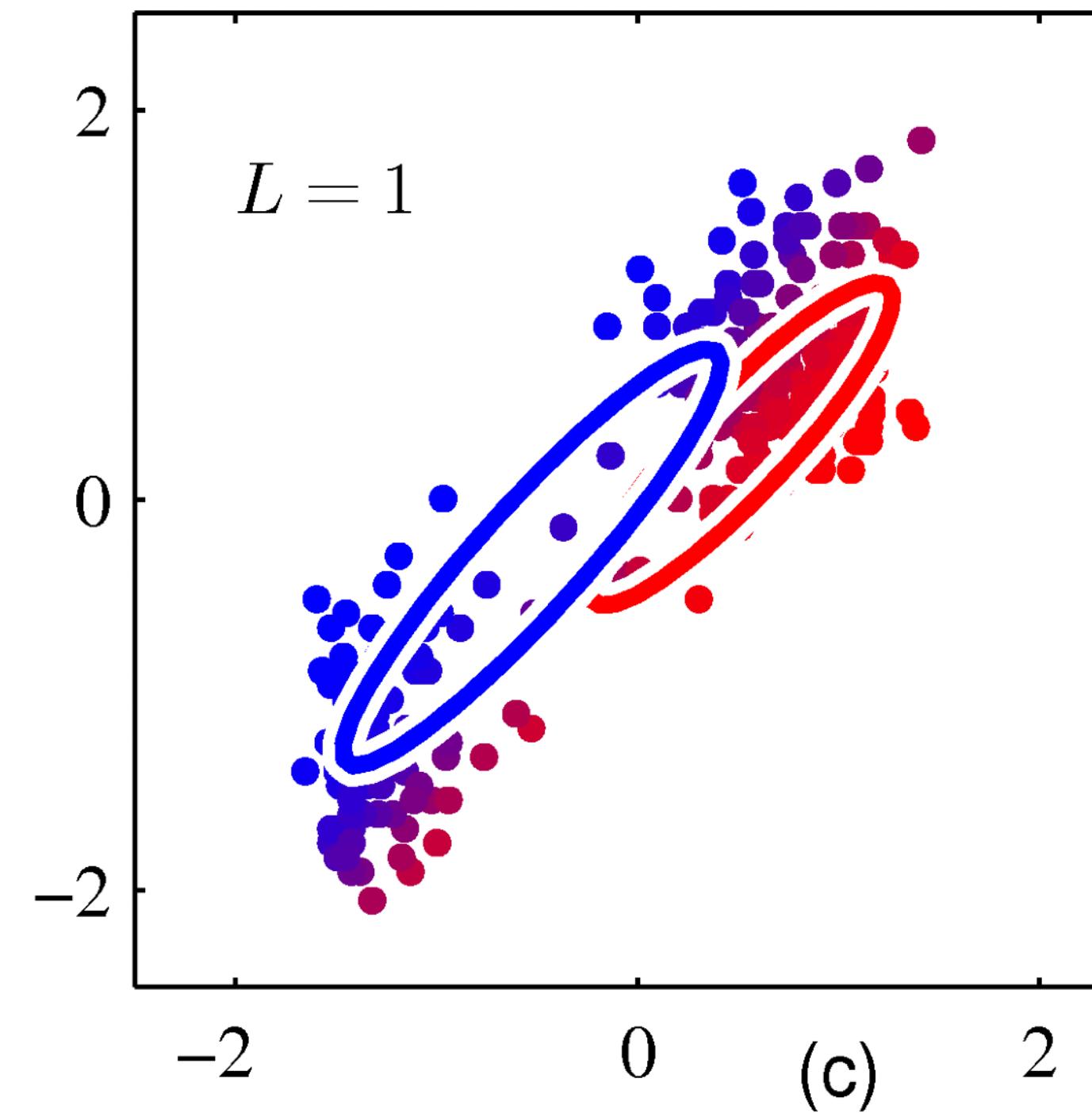


(a)

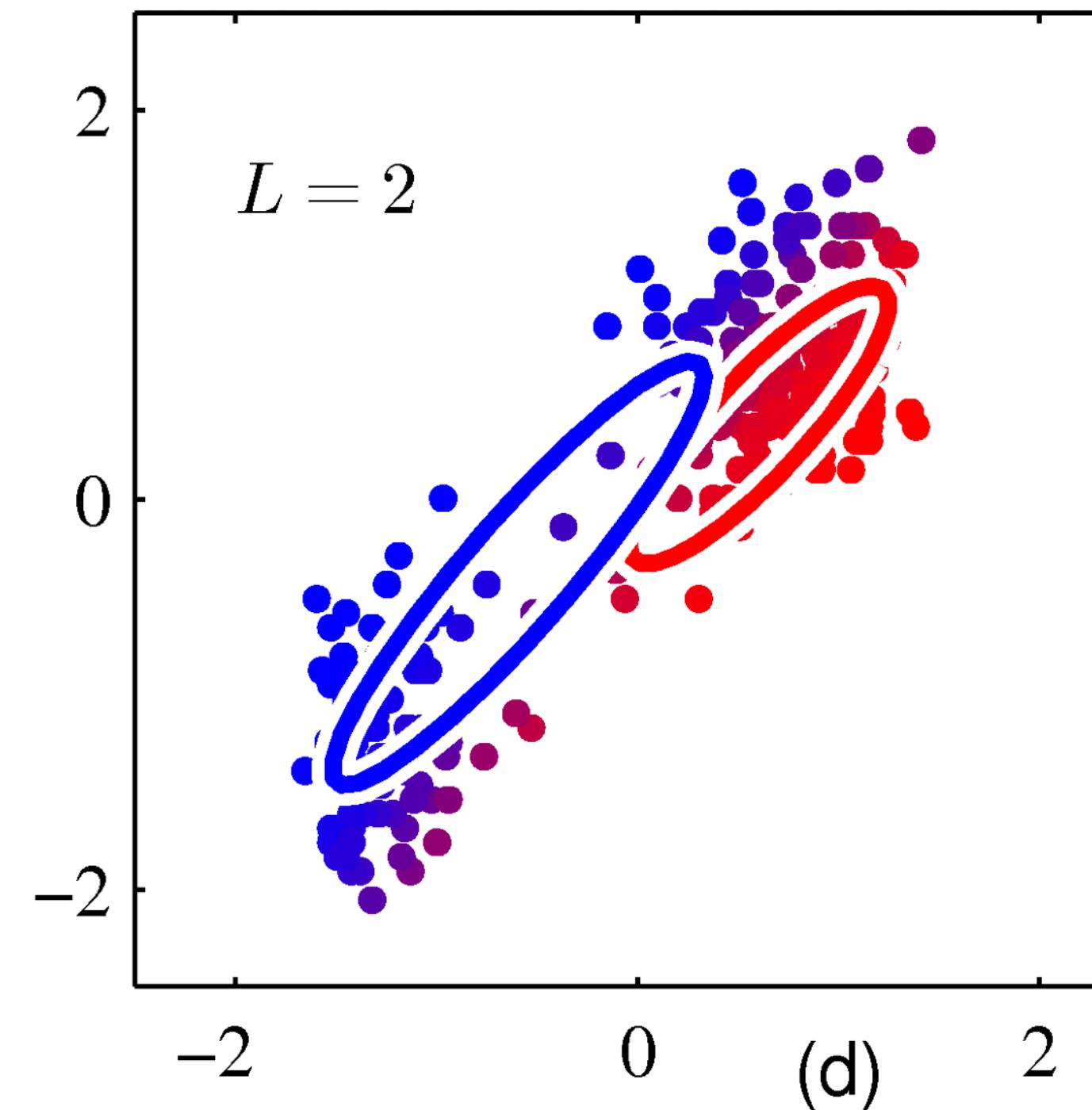
Clustering: Color Points according to X



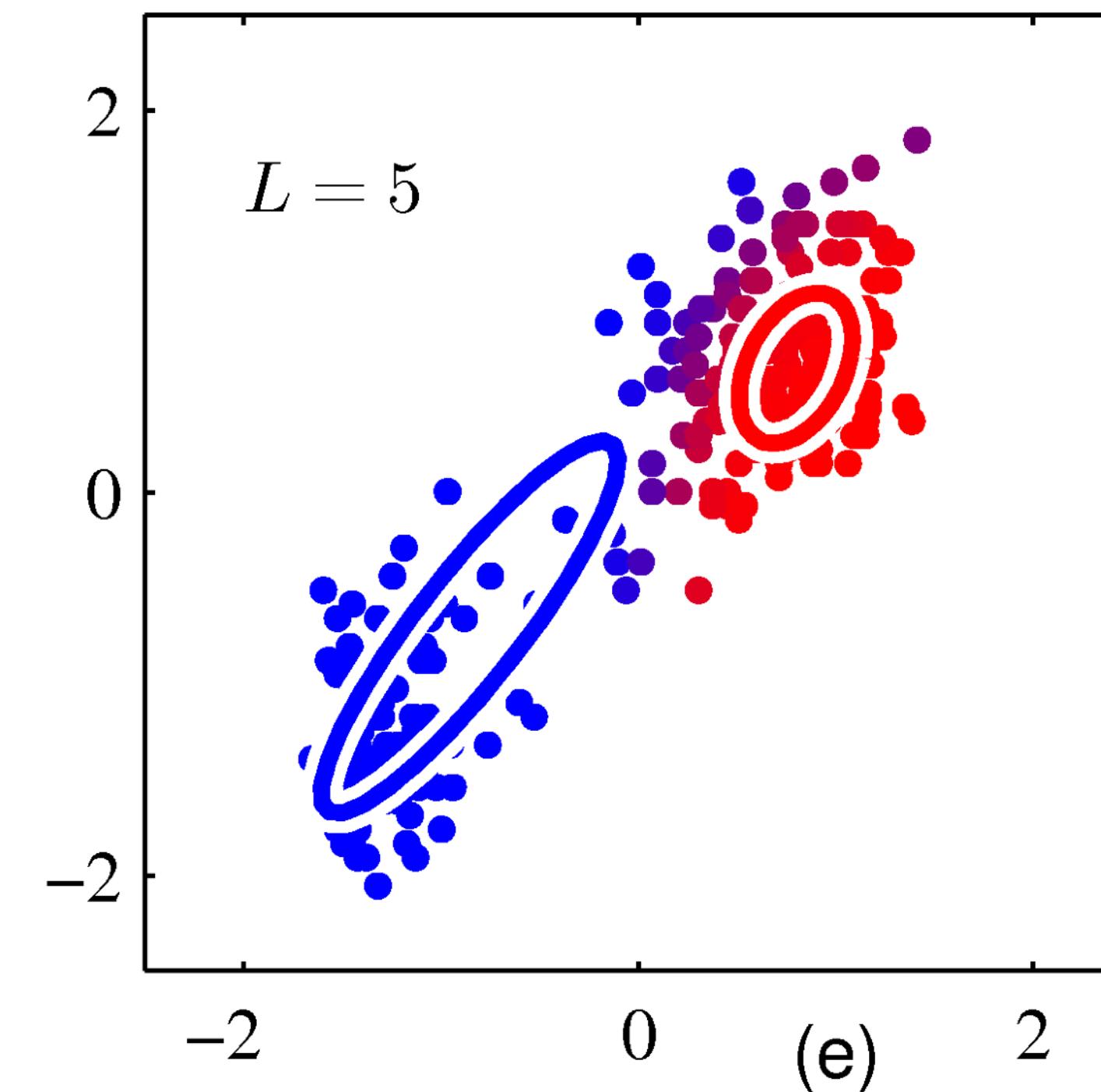
Clustering: Color Points according to X



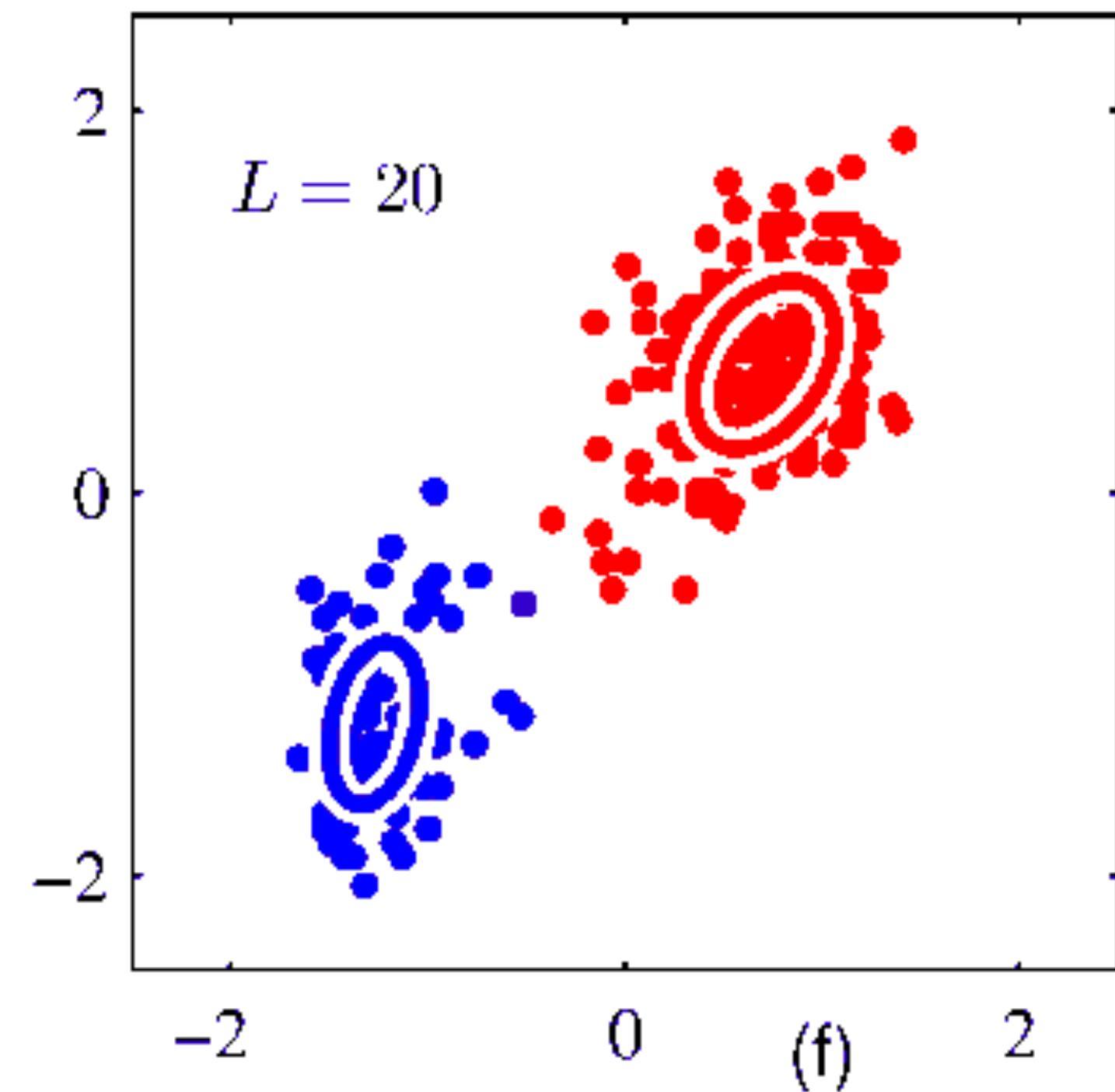
Clustering: Color Points according to X



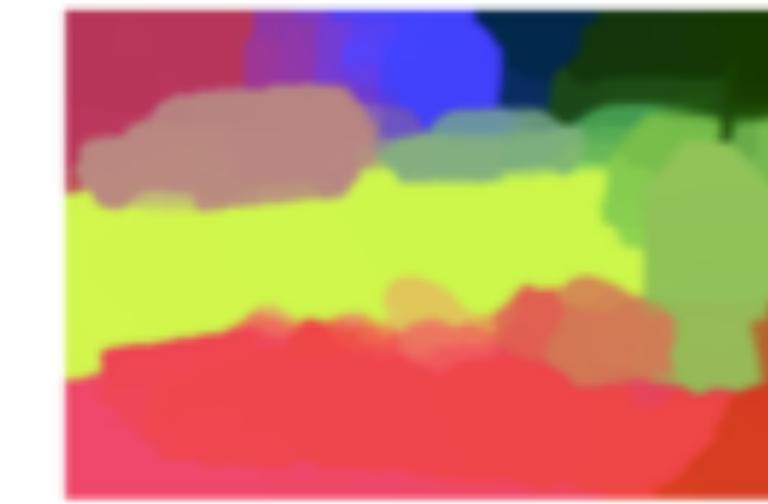
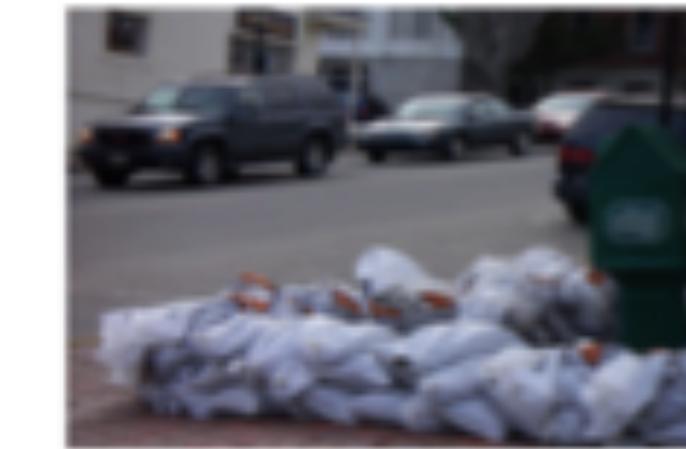
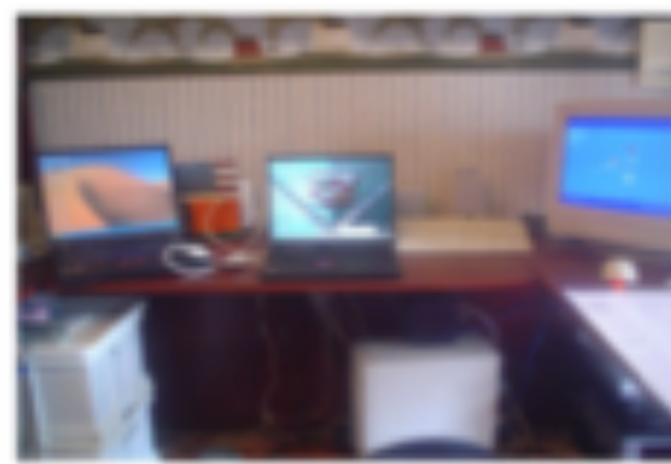
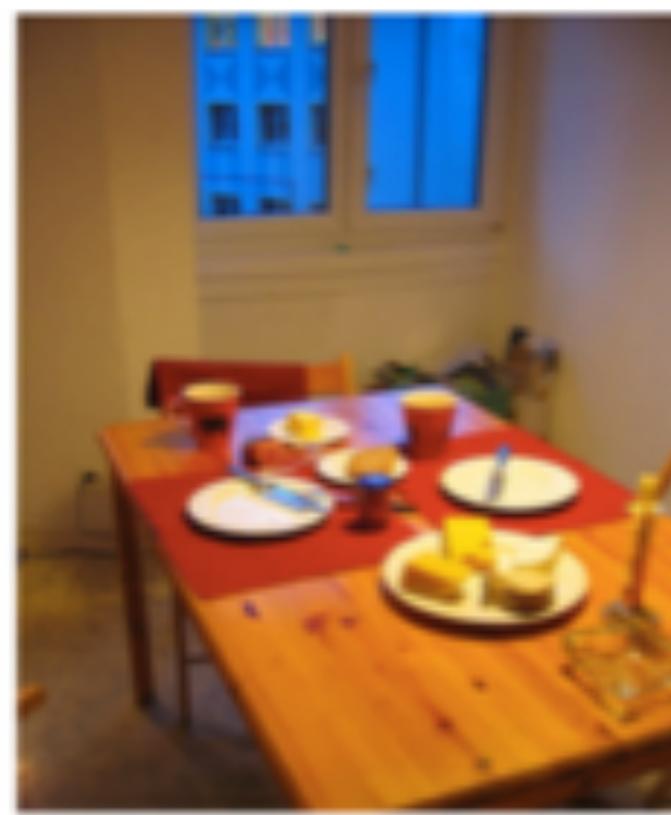
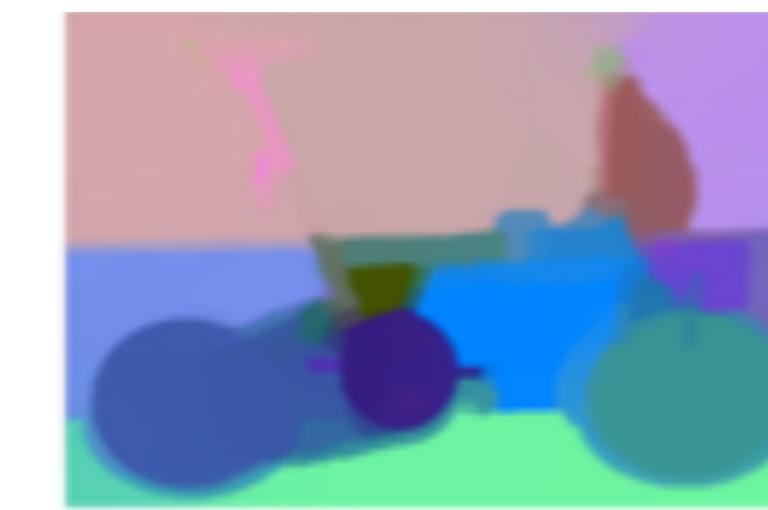
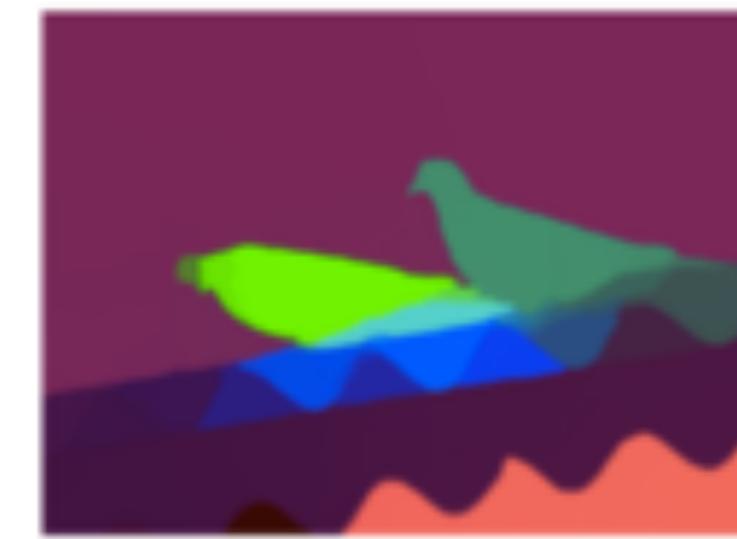
Clustering: Color Points according to X



Clustering: Color Points according to X



Clustering Examples: Image Segmentation using NCuts



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

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



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



$\bar{\mathbf{x}}$

Moving Along Learned Face Manifold

Male → Female



Female → Male

Trajectory along the “male” dimension

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

Moving Along Learned Face Manifold

Male → Female



Female → Male

Trajectory along the “male” dimension

Young → Old



Old → Young

Trajectory along the “young” dimension
[Lample et. al. Fader Networks, NeurIPS 2017]

Image Classification



$$\mathbb{R}^{3WH} \rightarrow \mathbb{Z}$$

Image Classification



$$\mathbb{R}^{3WH} \rightarrow \mathbb{Z}$$

Human
Cat
Dog
Tiger
Chair

Image Classification



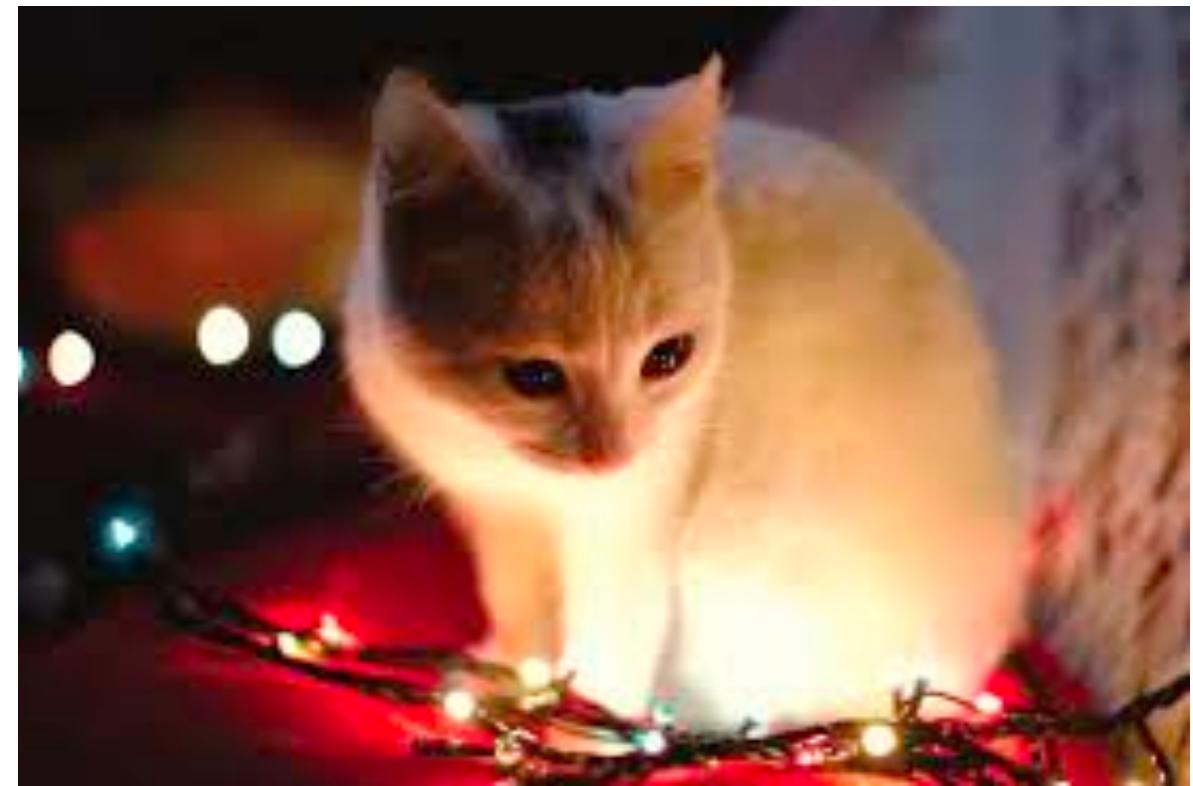
$$\mathbb{R}^{3WH} \rightarrow \mathbb{Z}$$

**Human
Cat
Dog
Tiger
Chair**

How will random guess or ‘chance’ fair?

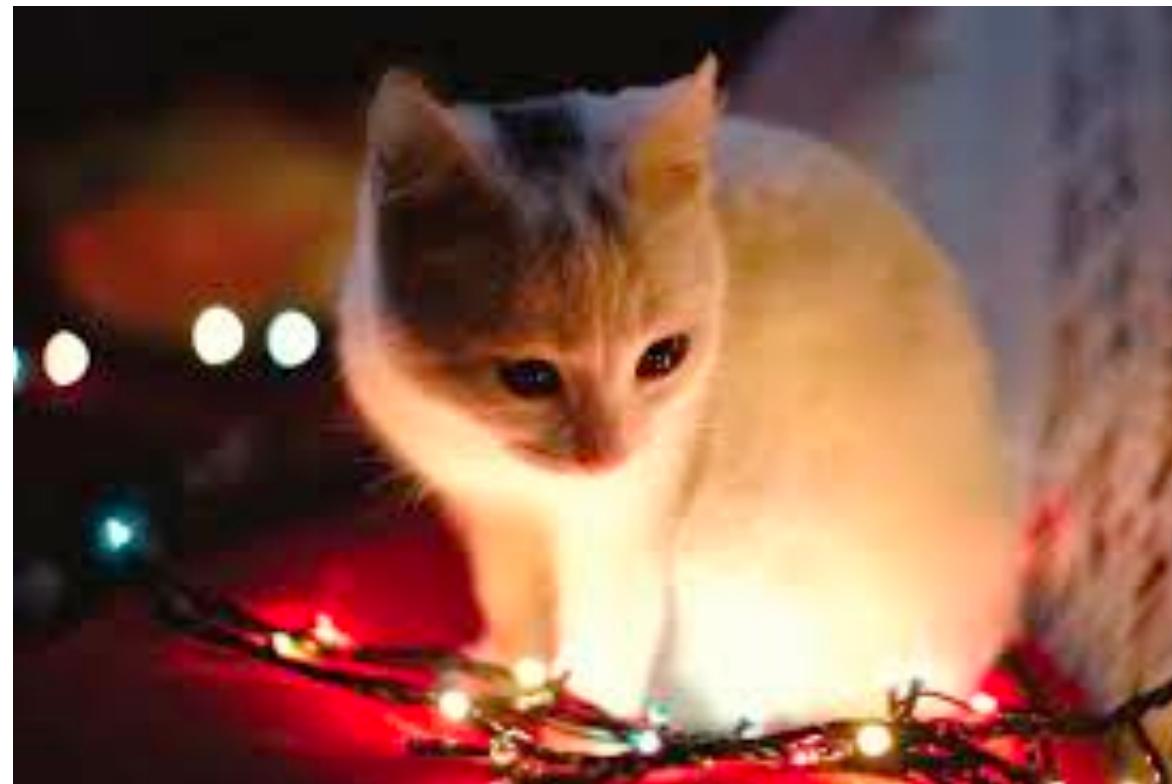
Challenges

Challenges

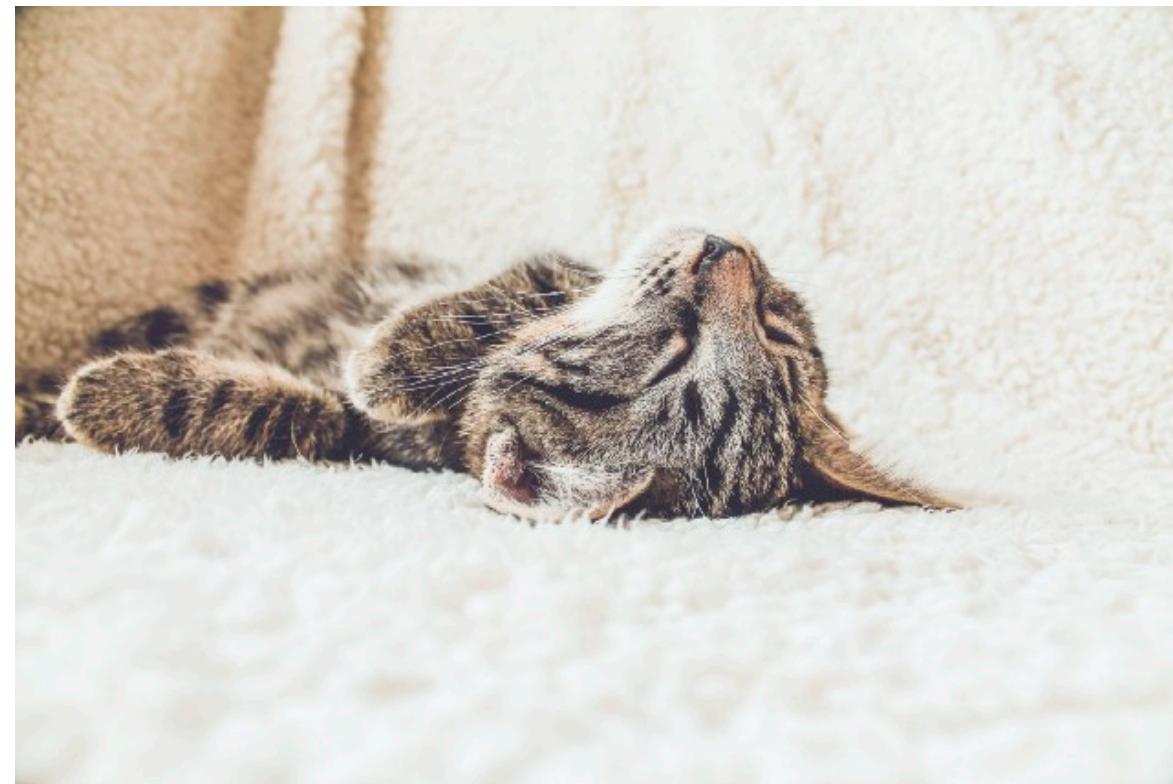


Illumination

Challenges

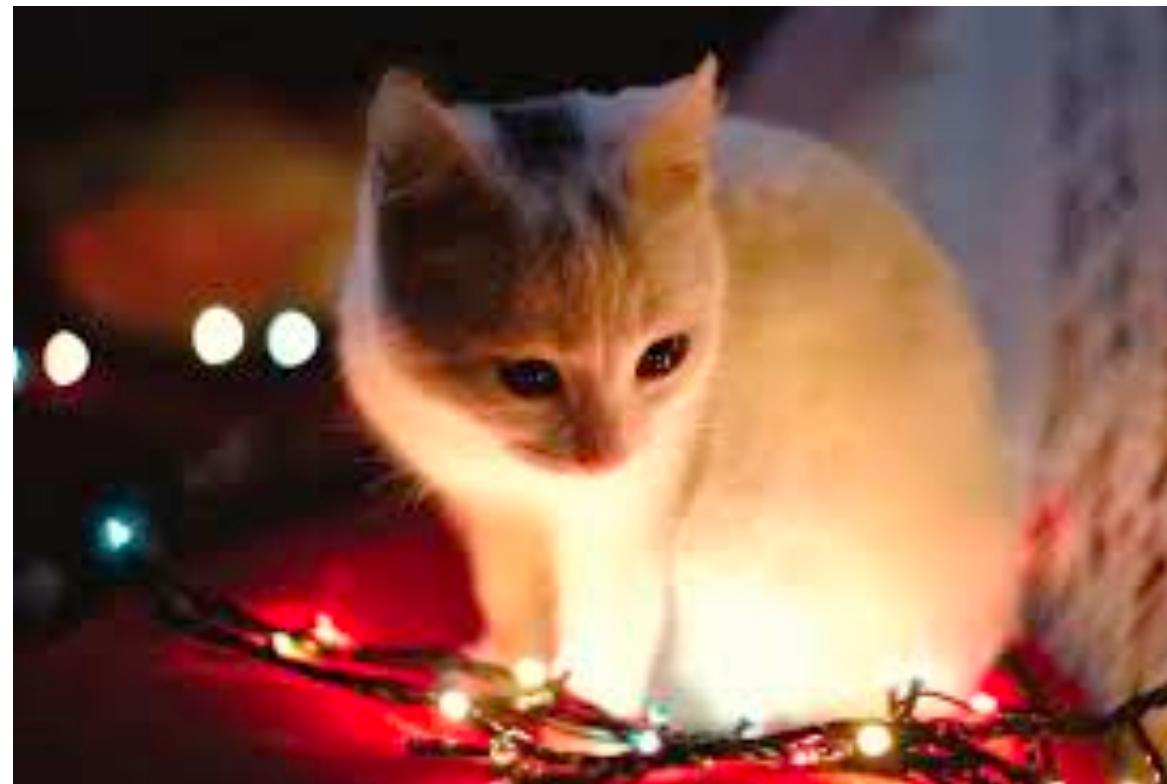


Illumination



Occlusion

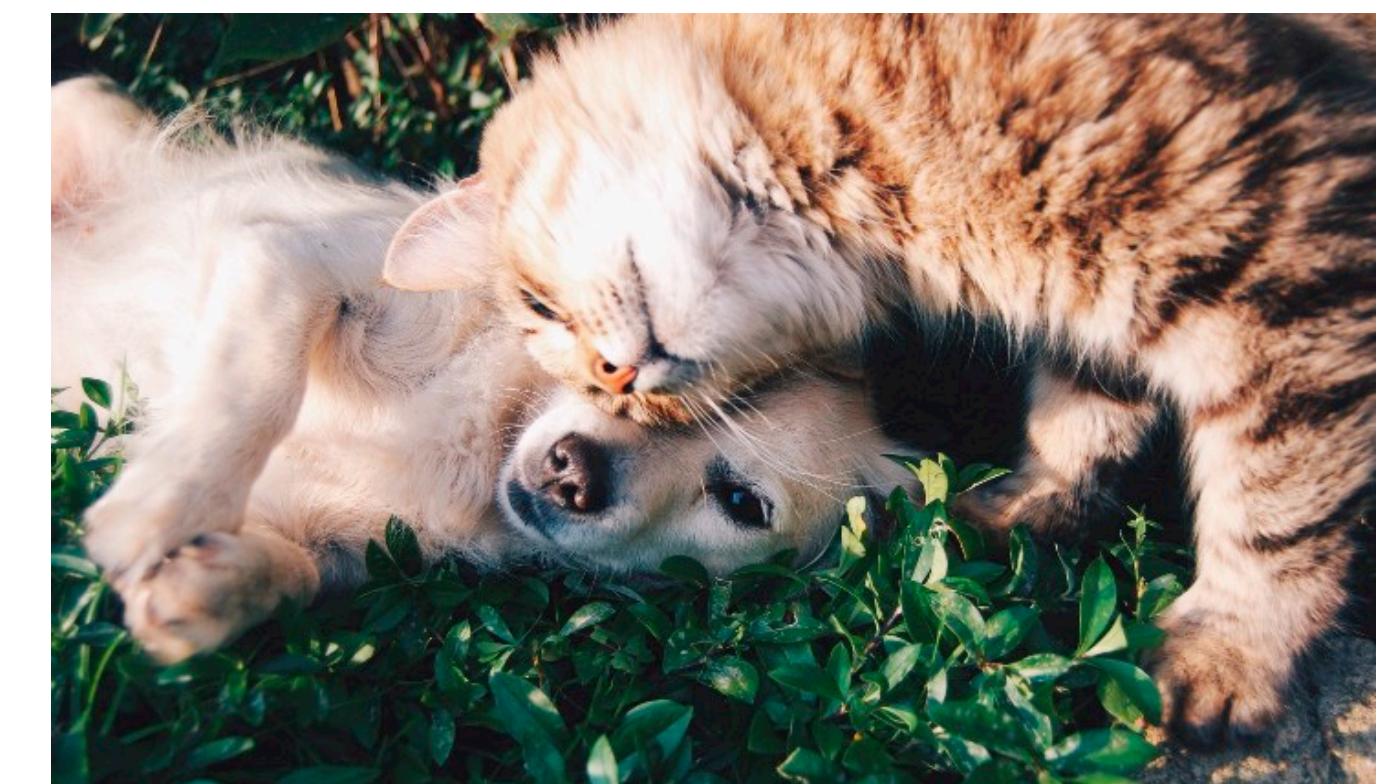
Challenges



Illumination

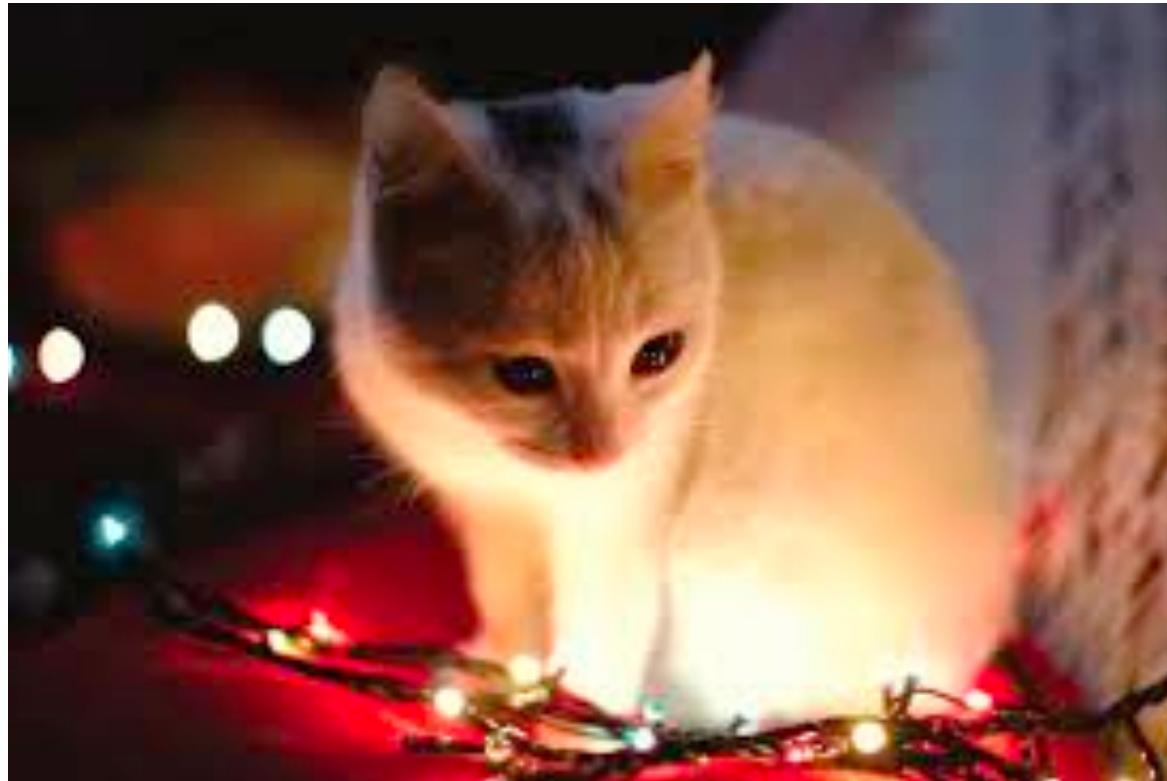


Occlusion



Camera/object angle

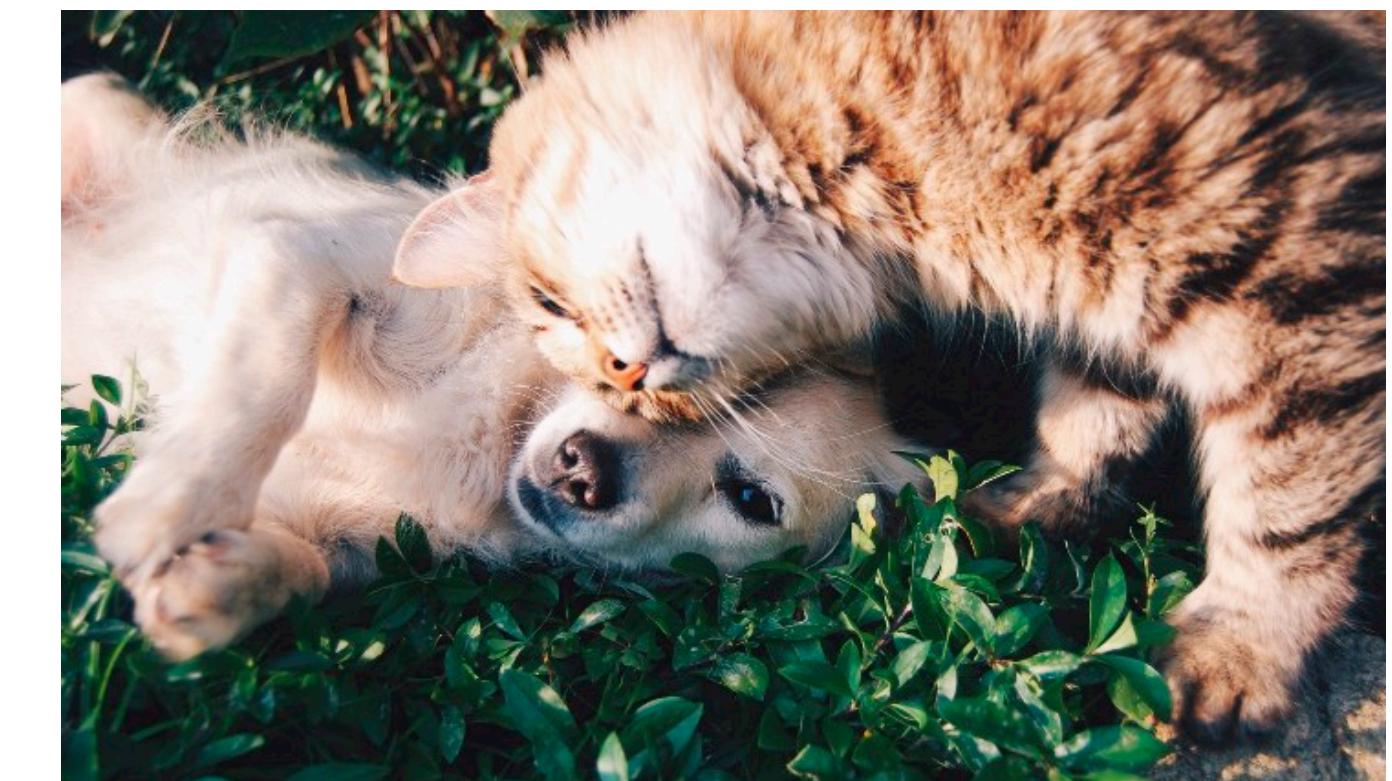
Challenges



Illumination



Occlusion



Camera/object angle



Deformation

Challenges



Illumination



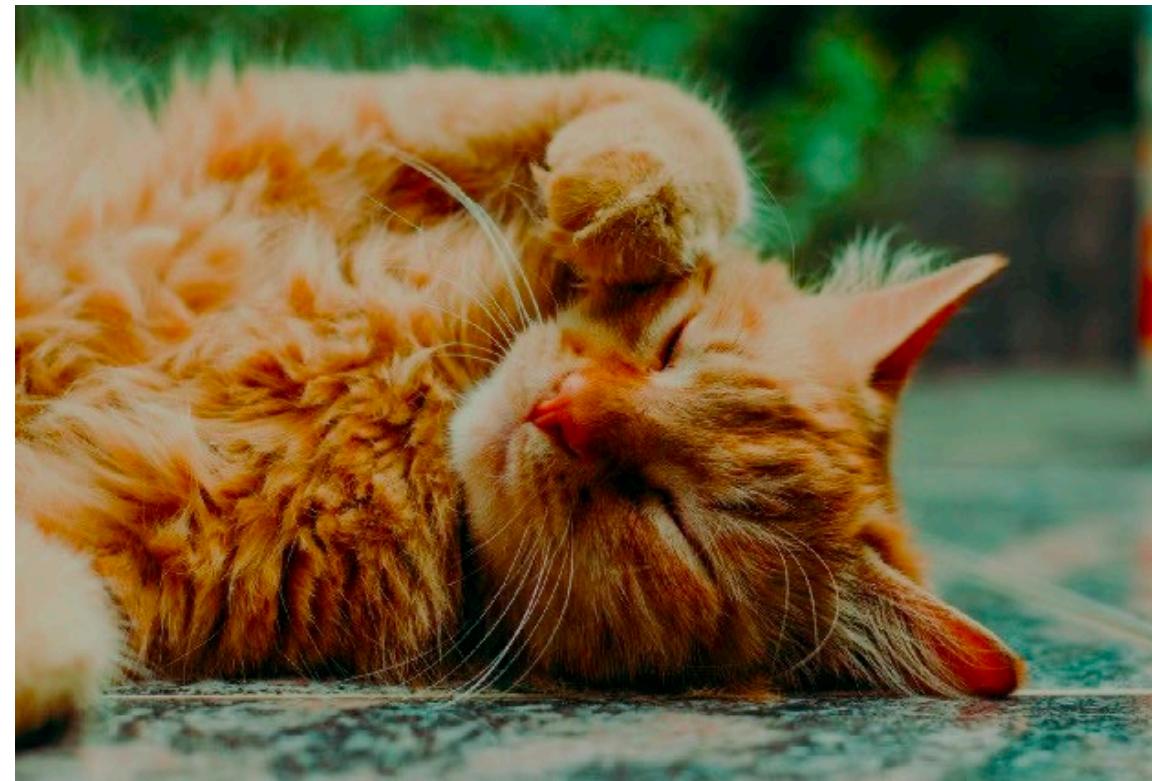
Occlusion



Camera/object angle

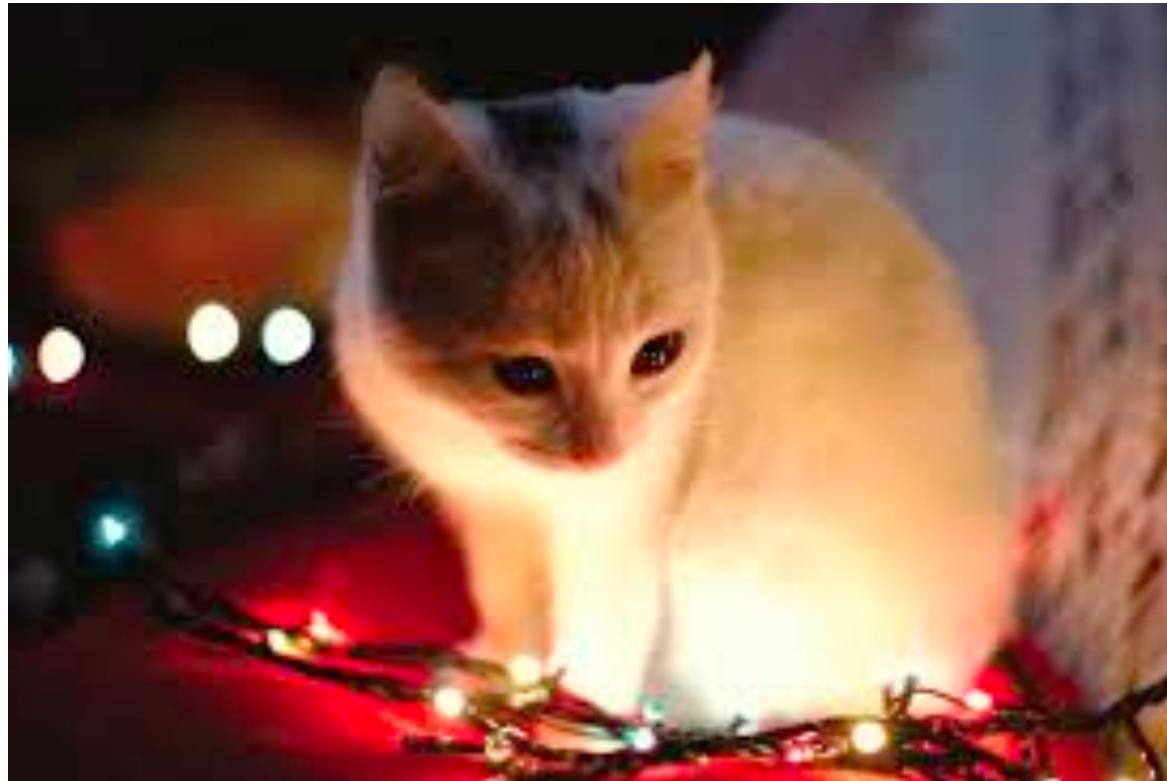


Deformation



Color balance

Challenges



Illumination



Occlusion



Camera/object angle



Deformation

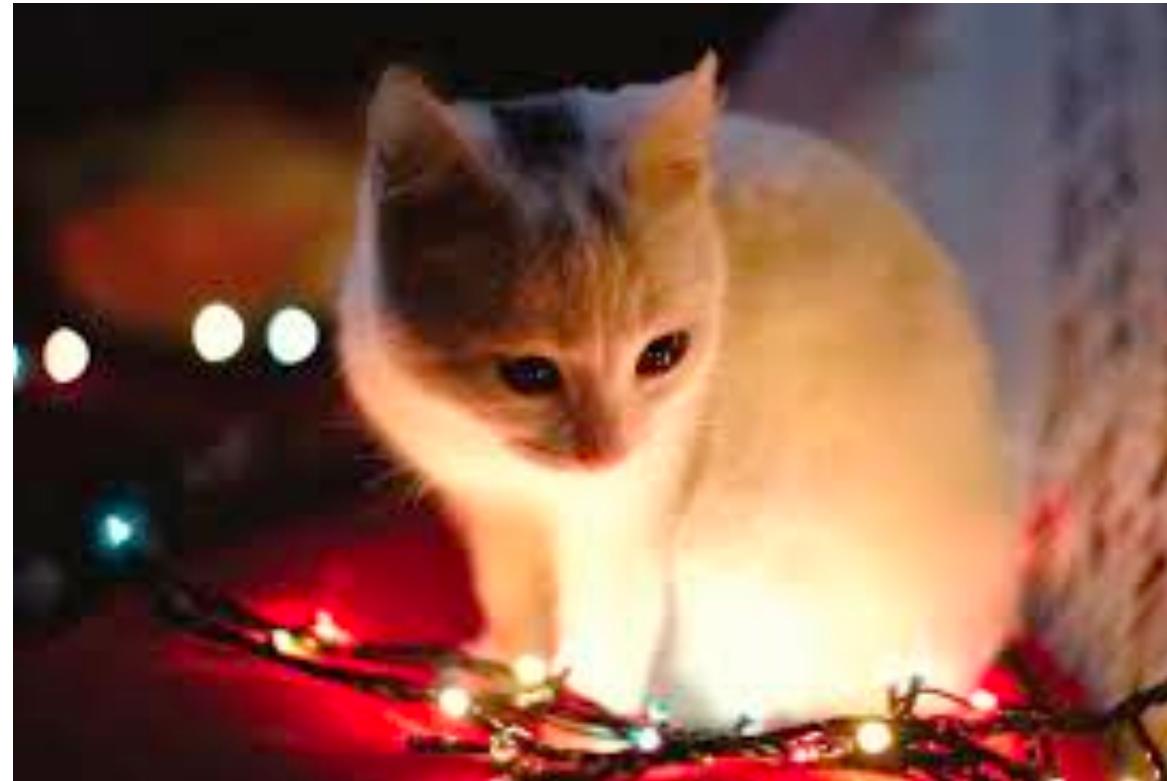


Color balance



Background

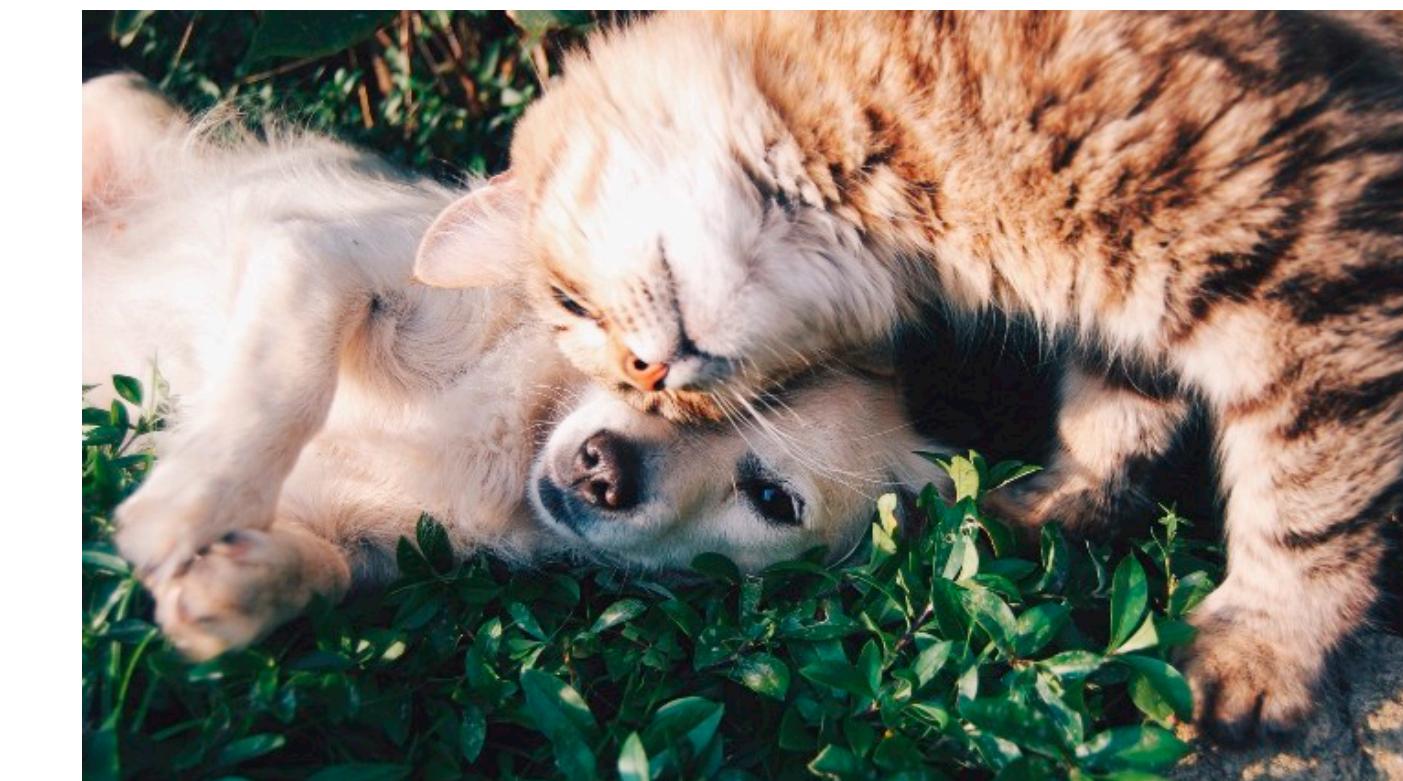
Challenges



Illumination



Occlusion



Camera/object angle



Deformation



Color balance



Background

Nearest Neighbor Classifier

- **Training data** $X := \{(I_1, l_1), (I_2, l_2), \dots, (I_n, l_n)\}$

$$I_j \in \mathbb{R}^{r \times c \times 3} \quad l_j \in \mathbb{Z}^k$$

- **Test data**

$$I_q$$

What class does this belong to?

How to compare two images?

$$D(I_1, I_2) := \sum_{i,j} (I_1(i, j) - I_2(i, j))^2$$

How to compare two images?

$$D(I_1, I_2) := \sum_{i,j} (I_1(i, j) - I_2(i, j))^2$$

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167

How to compare two images?

$$D(I_1, I_2) := \sum_{i,j} (I_1(i, j) - I_2(i, j))^2$$

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167

-

37	8	14	200
20	140	70	35
15	1	195	100
9	99	55	170

How to compare two images?

$$D(I_1, I_2) := \sum_{i,j} (I_1(i, j) - I_2(i, j))^2$$

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167

-

37	8	14	200
20	140	70	35
15	1	195	100
9	99	55	170

=

4	4	4	10000
100	144	64	25
1	0	25	36
9	1	9	9

How to compare two images?

$$D(I_1, I_2) := \sum_{i,j} (I_1(i, j) - I_2(i, j))^2$$

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167

-

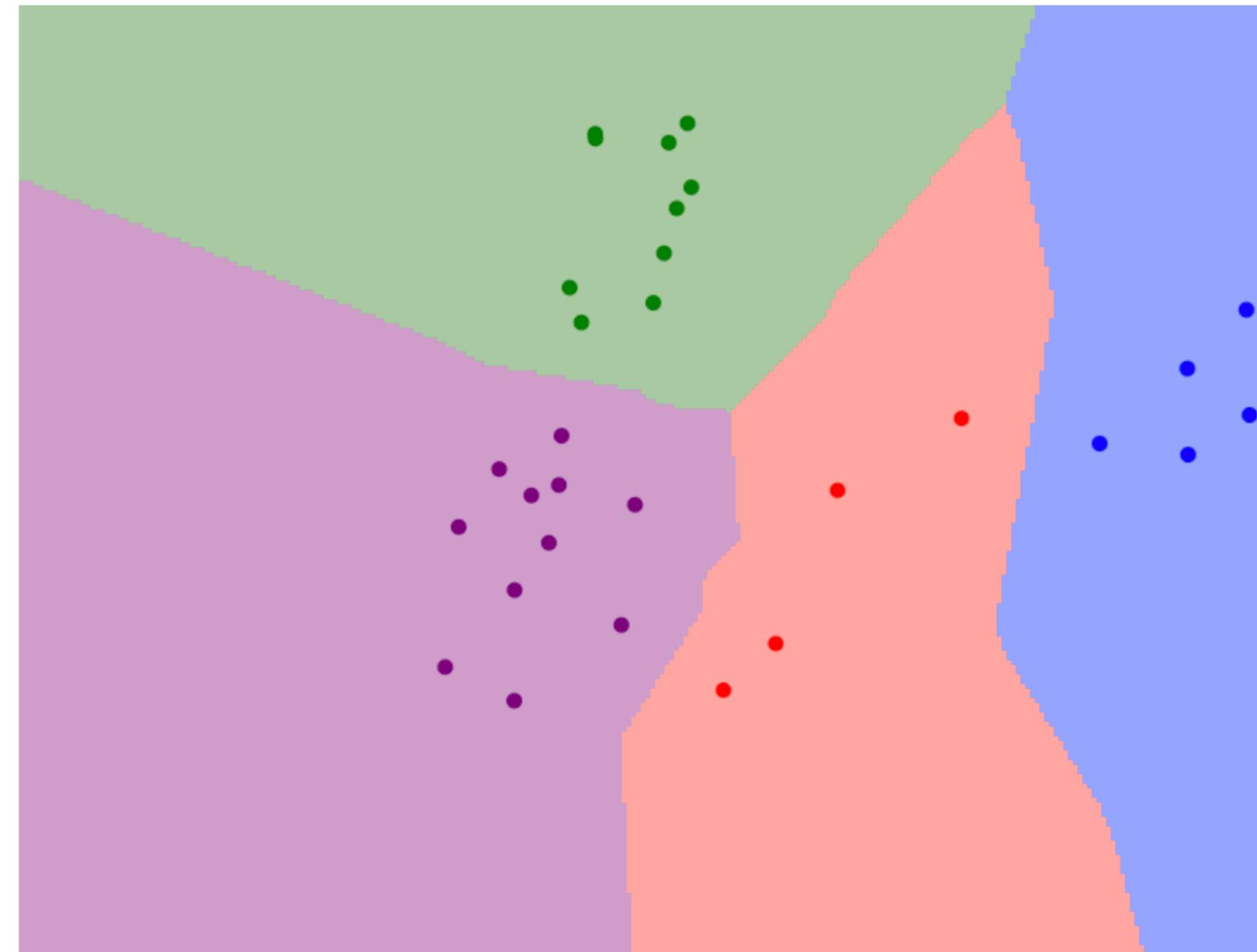
37	8	14	200
20	140	70	35
15	1	195	100
9	99	55	170

=

4	4	4	10000
100	144	64	25
1	0	25	36
9	1	9	9

10435

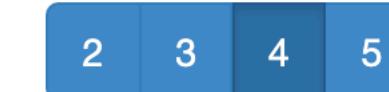
Nearest Neighbor Explained



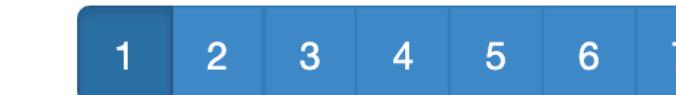
Metric



Num classes



Num Neighbors (K)

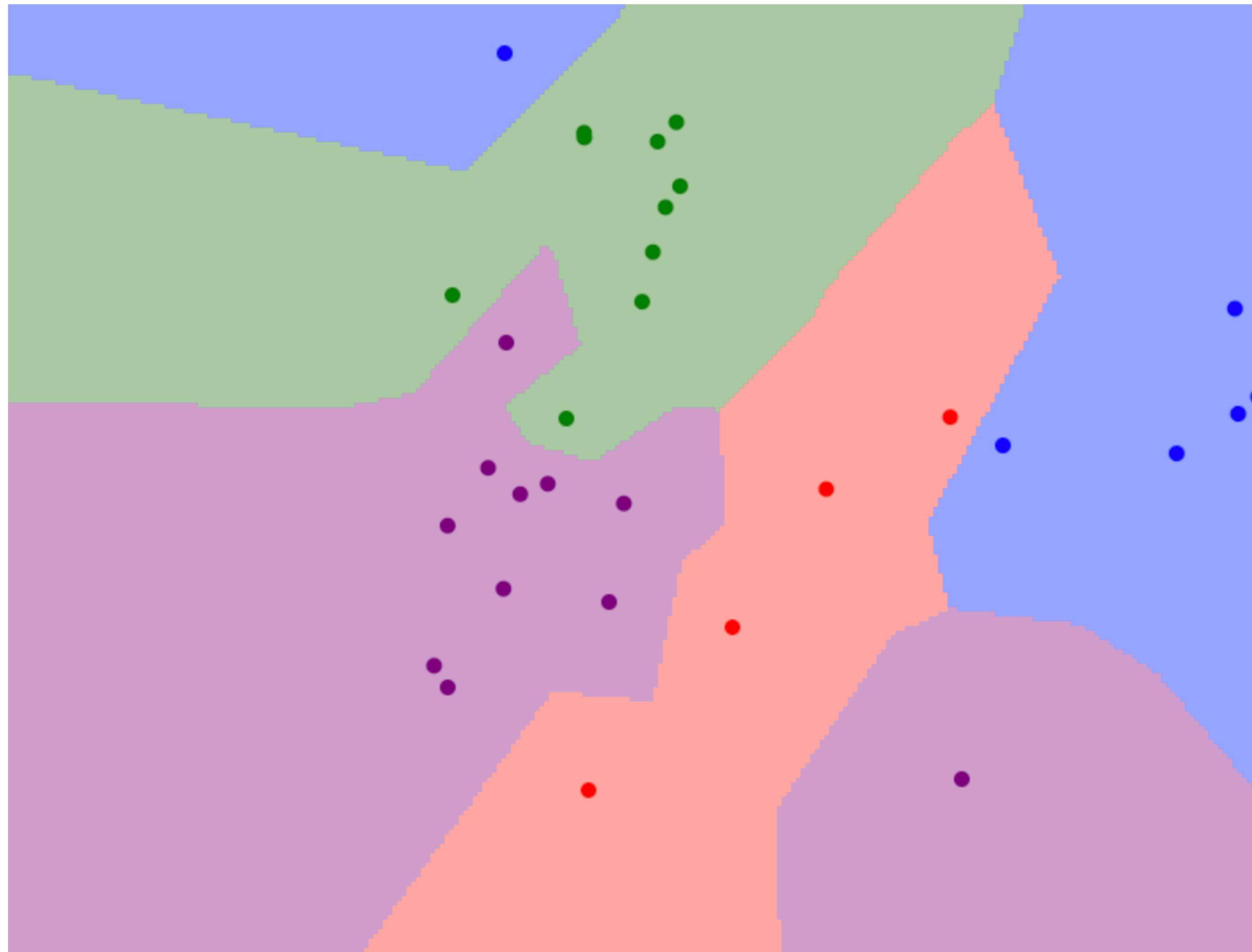


Num points

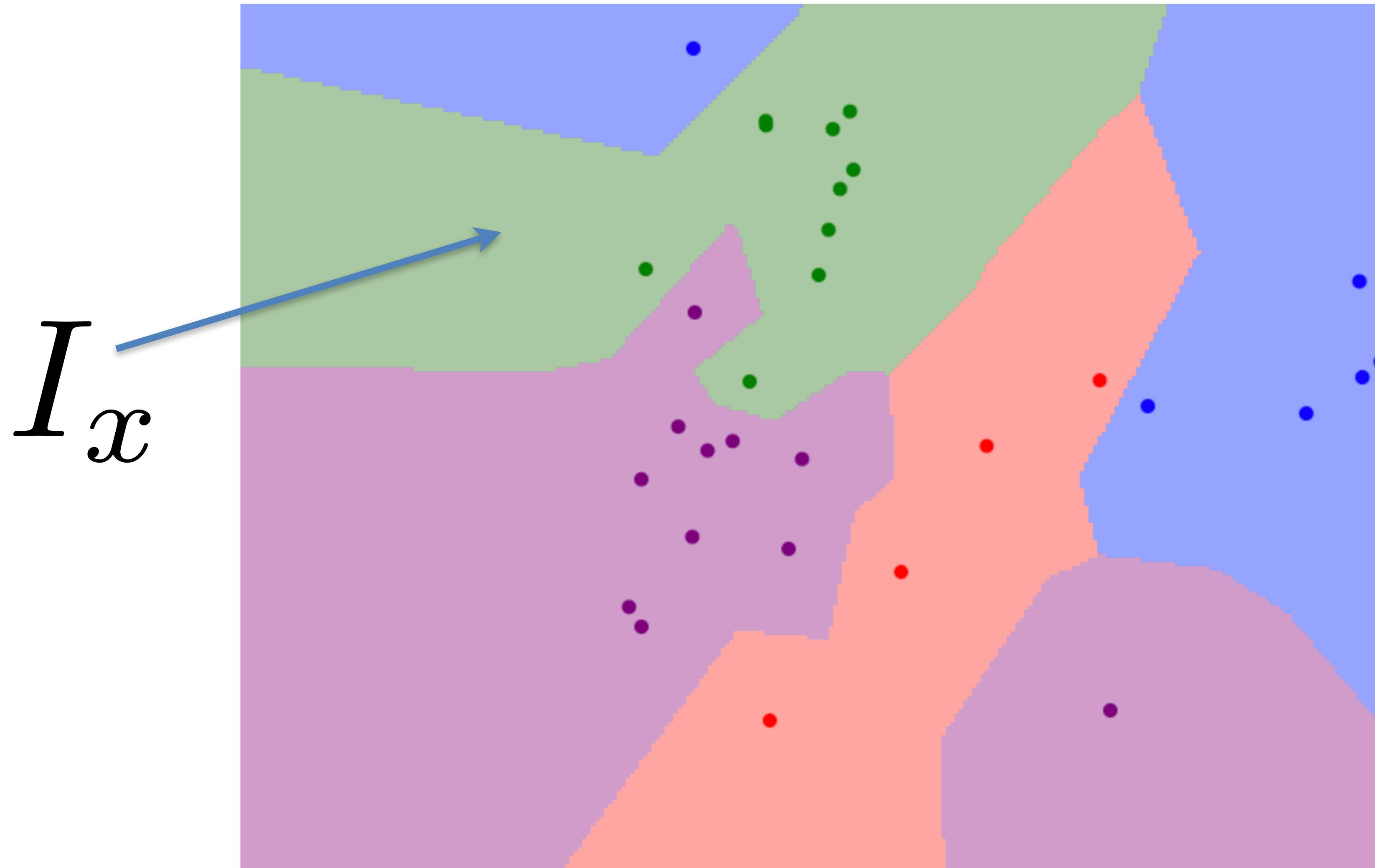


<http://vision.stanford.edu/teaching/cs231n-demos/knn/>

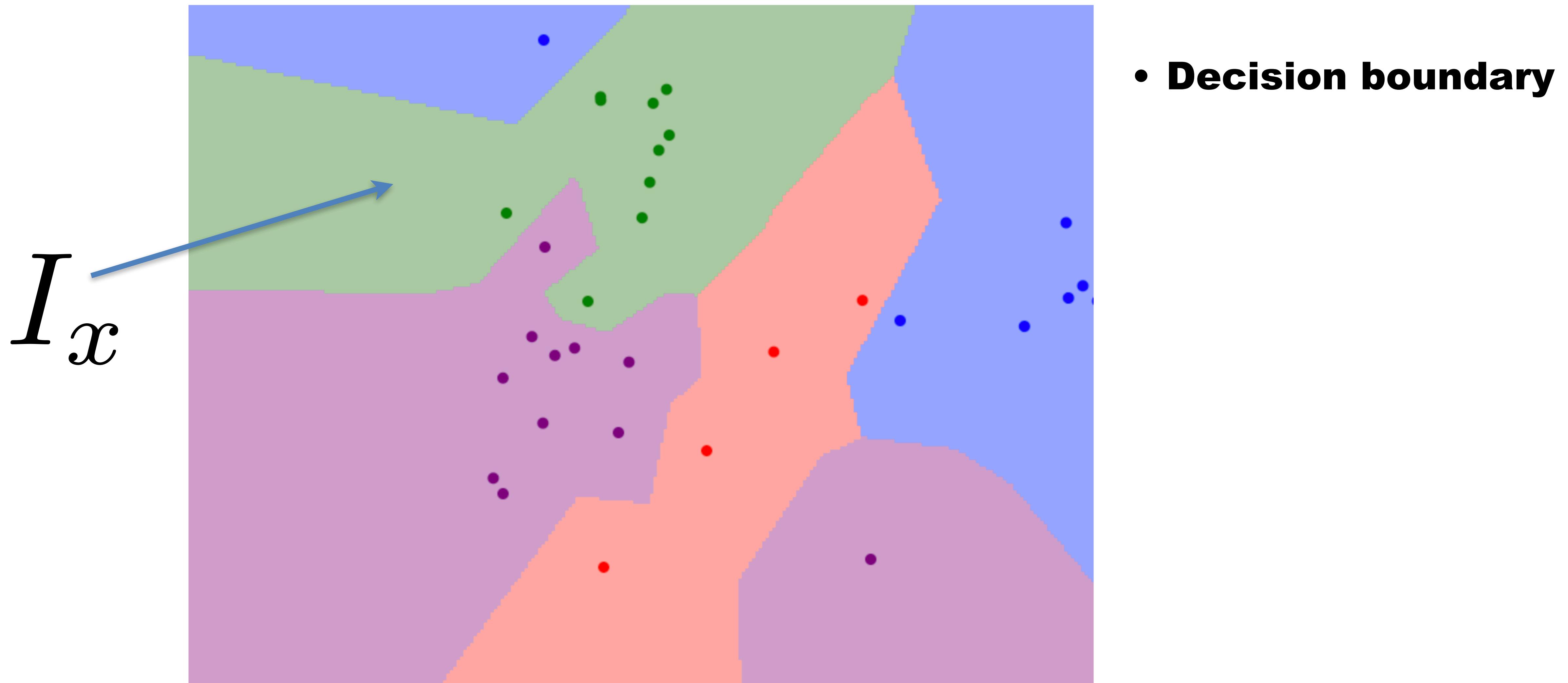
Nearest Neighbor Explained



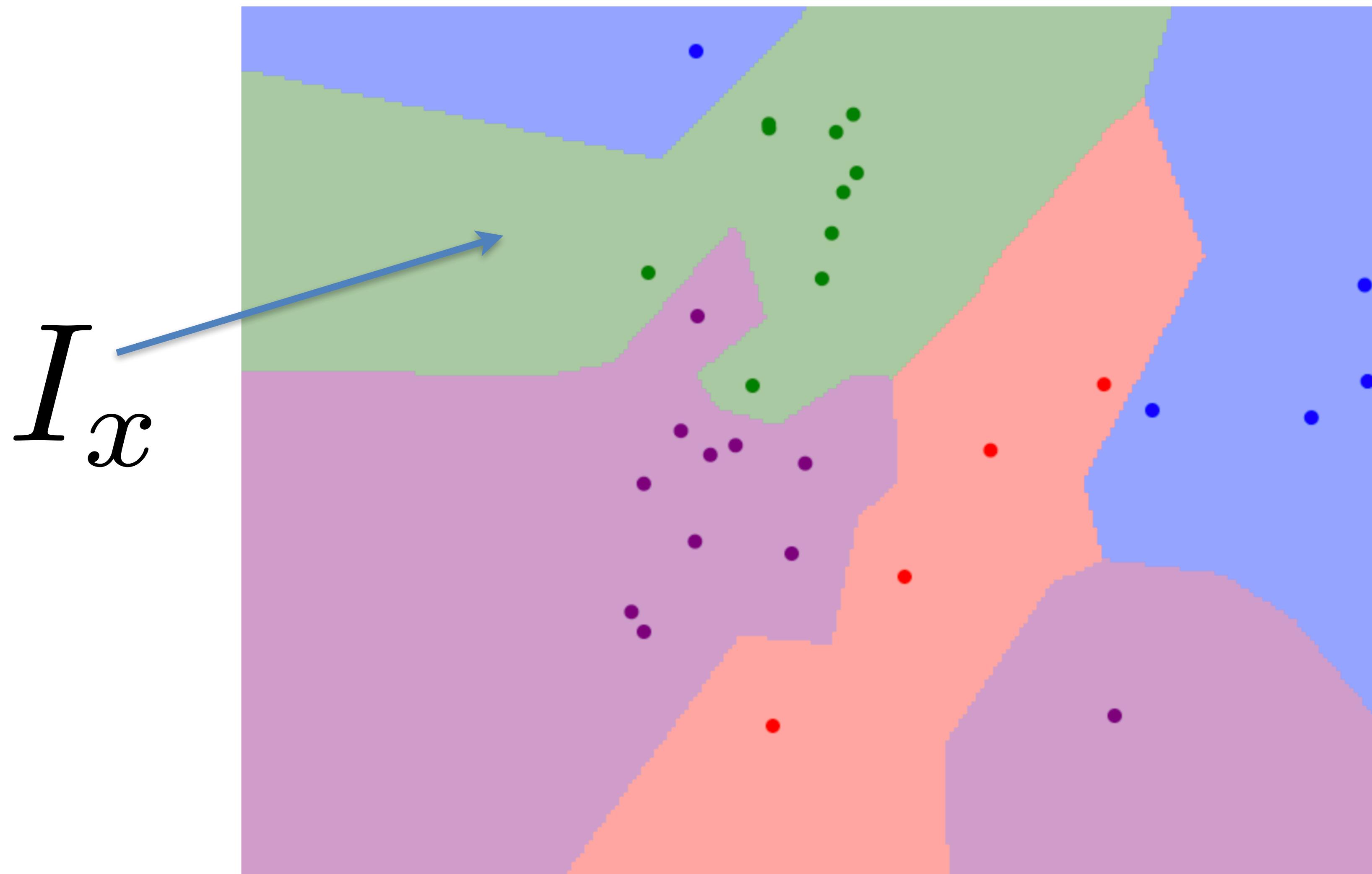
Nearest Neighbor Explained



Nearest Neighbor Explained

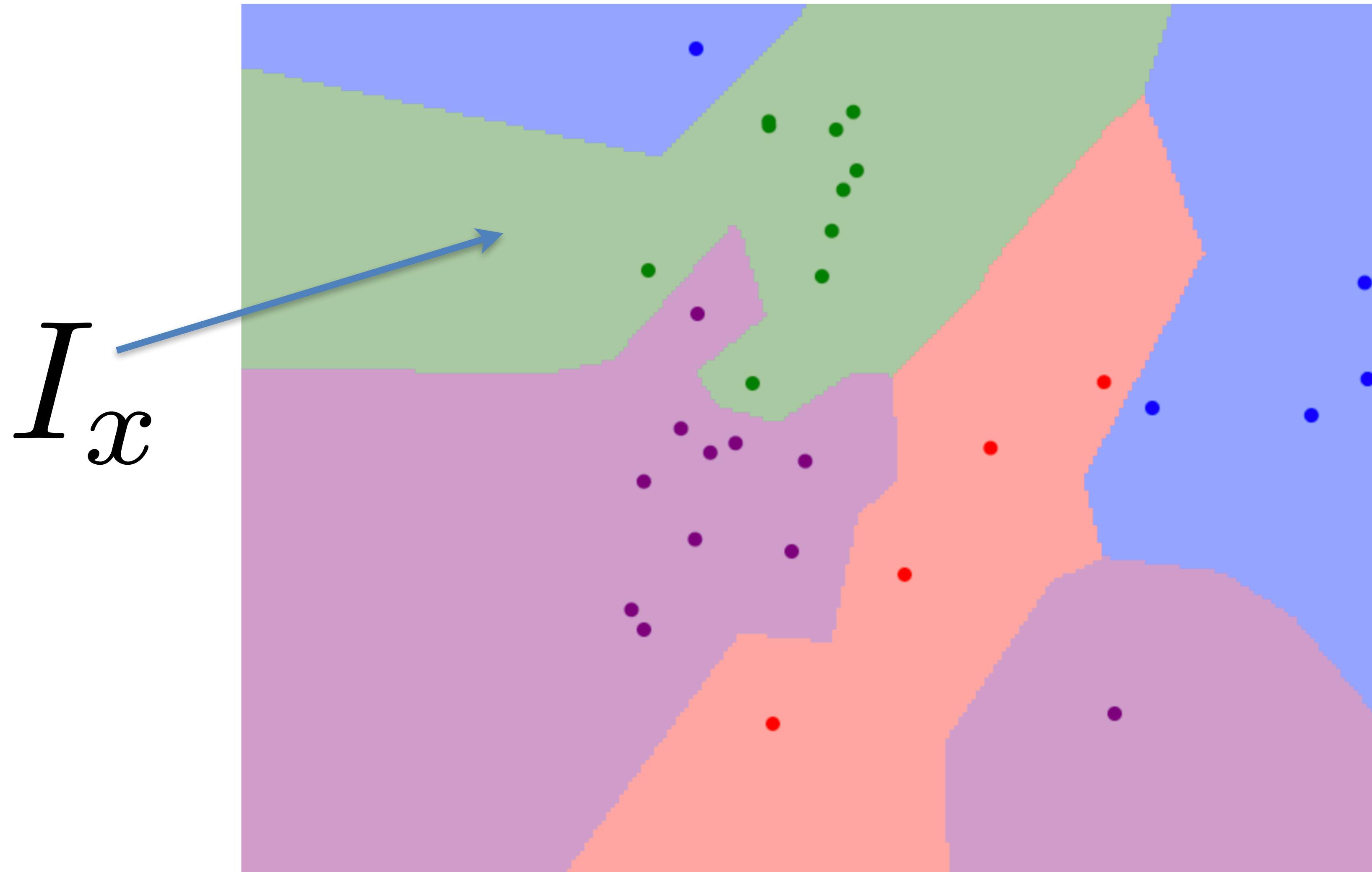


Nearest Neighbor Explained



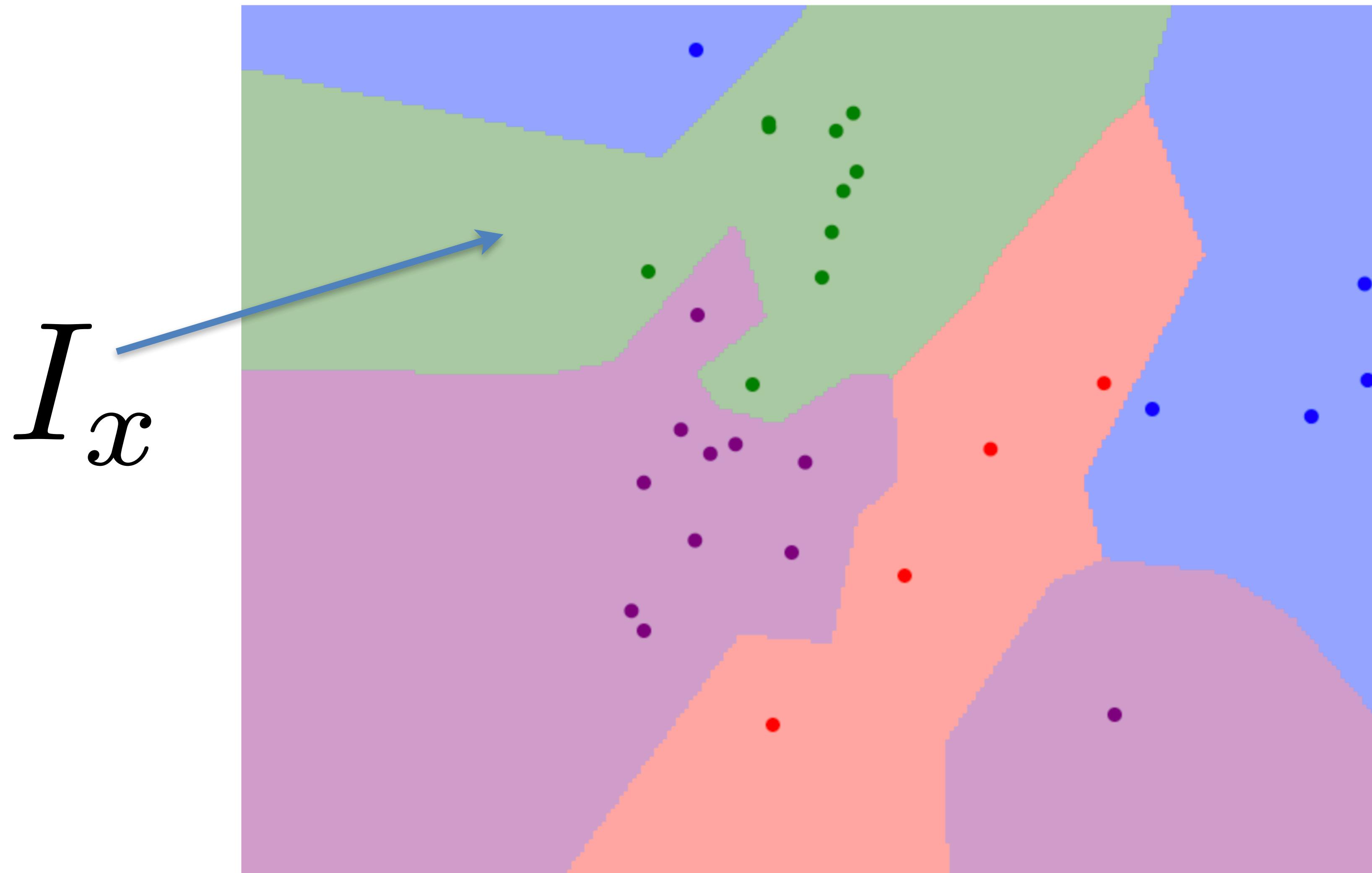
- **Decision boundary**
- **Jagged boundary**

Nearest Neighbor Explained



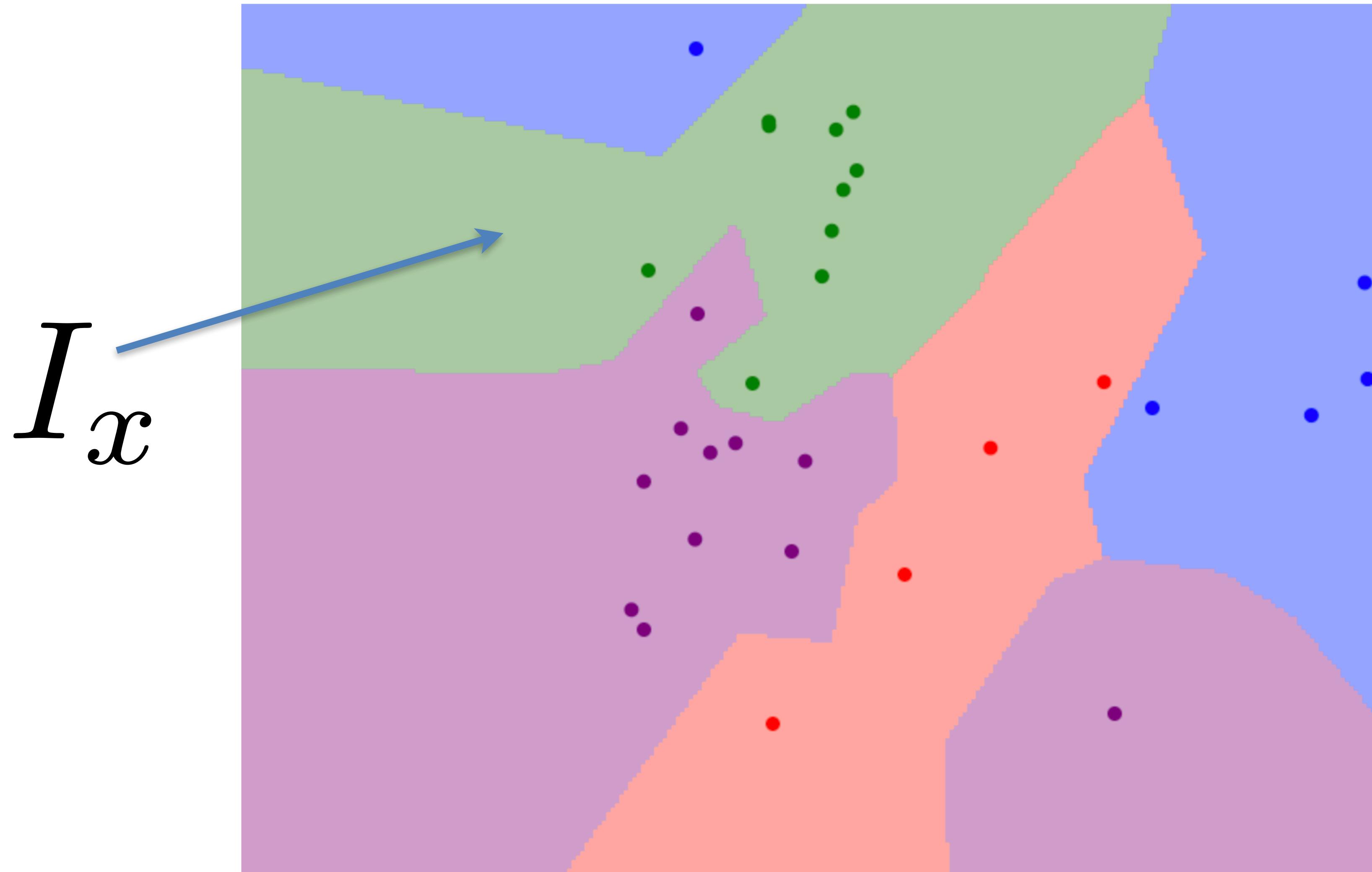
- **Decision boundary**
- **Jagged boundary**
- **Isolated regions**

Nearest Neighbor Explained



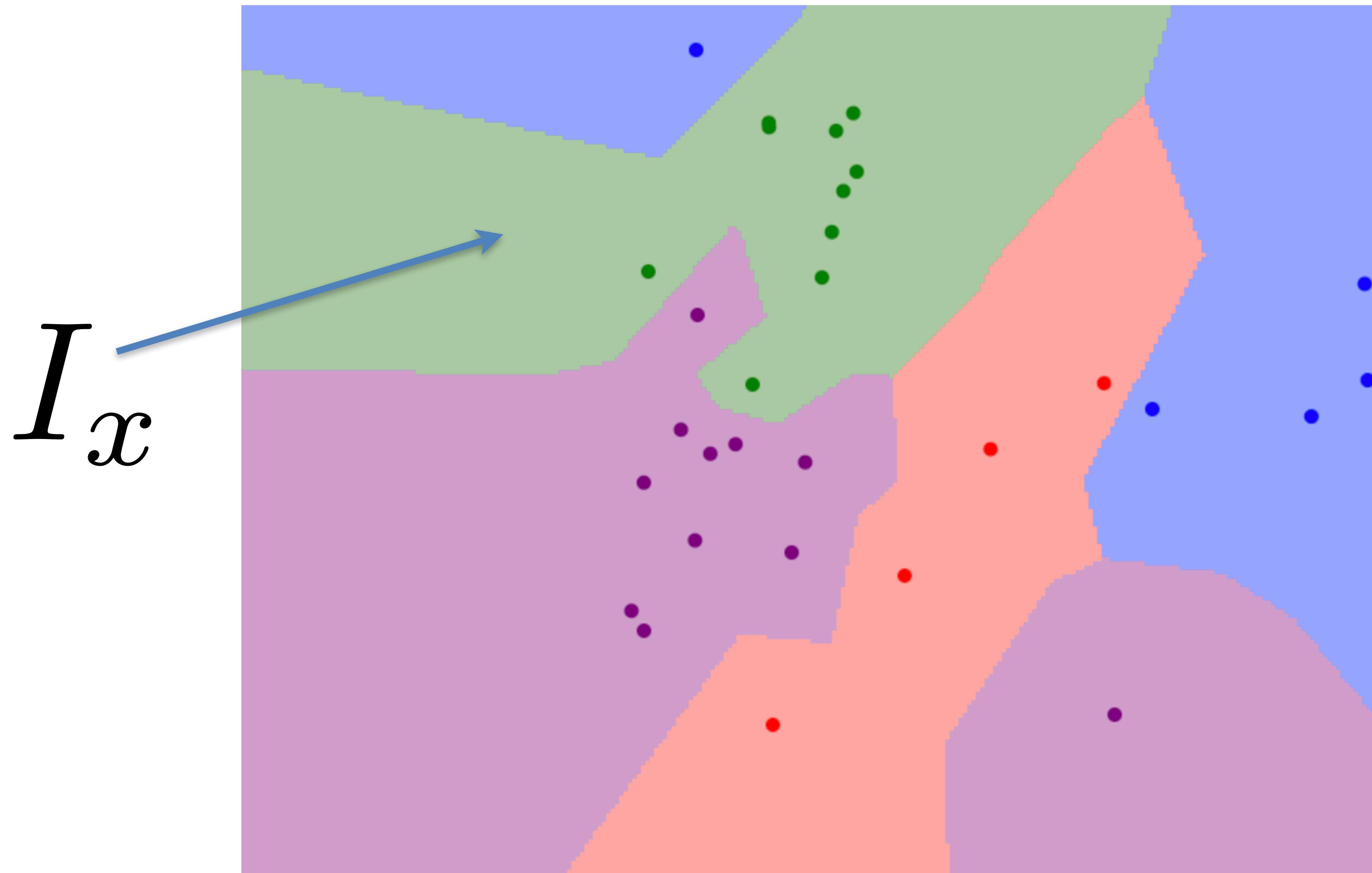
- **Decision boundary**
- **Jagged boundary**
- **Isolated regions**
- **Error prone**

Nearest Neighbor Explained



- **Decision boundary**
- **Jagged boundary**
- **Isolated regions**
- **Error prone**
- **Training time $O(1)$**

Nearest Neighbor Explained



- **Decision boundary**
- **Jagged boundary**
- **Isolated regions**
- **Error prone**
- **Training time $O(1)$**
- **Test time $O(n)$**

Dataset: MNIST

- **10 classes**
- **28x28**
- **50k/10k (training/test)**



Dataset: CIFAR10

- 10 classes
- 32x32
- 50k/10k (training/test)



Dataset: CIFAR100

- 100 classes
 - 32x32
 - 50k/10k (training/test)
 - 20 superclasses
(fine-grained classification)



Dataset: ImageNet

- **1000 classes**
- **256x256 (?)**
- **14M/1M (training/test)**

- **Total number of non-empty synsets: 21841**
- **Total number of images: 14,197,122**
- **Number of images with bounding box annotations: 1,034,908**
- **Number of synsets with SIFT features: 1000**
- **Number of images with SIFT features: 1.2 million**

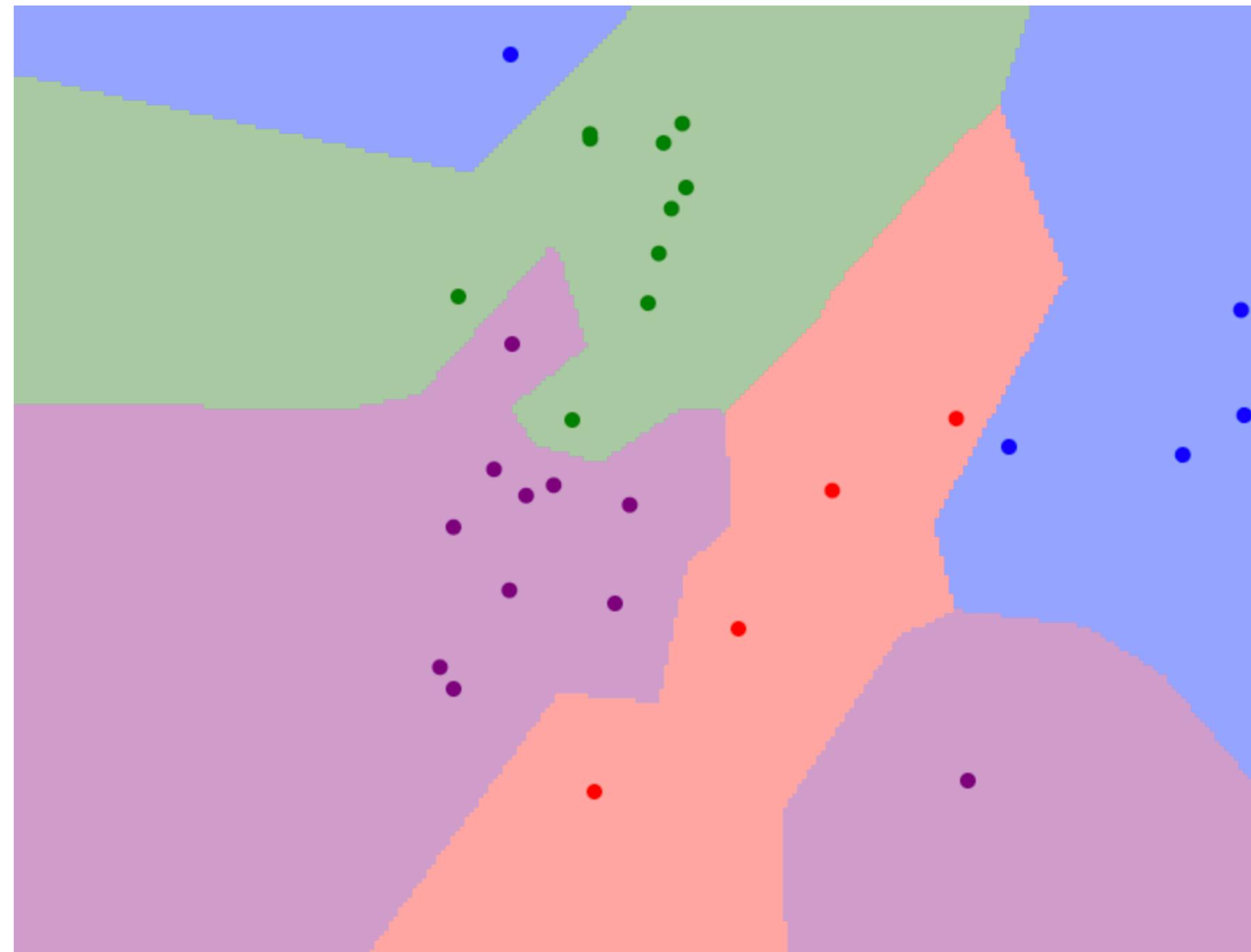


Dataset: MIT Places

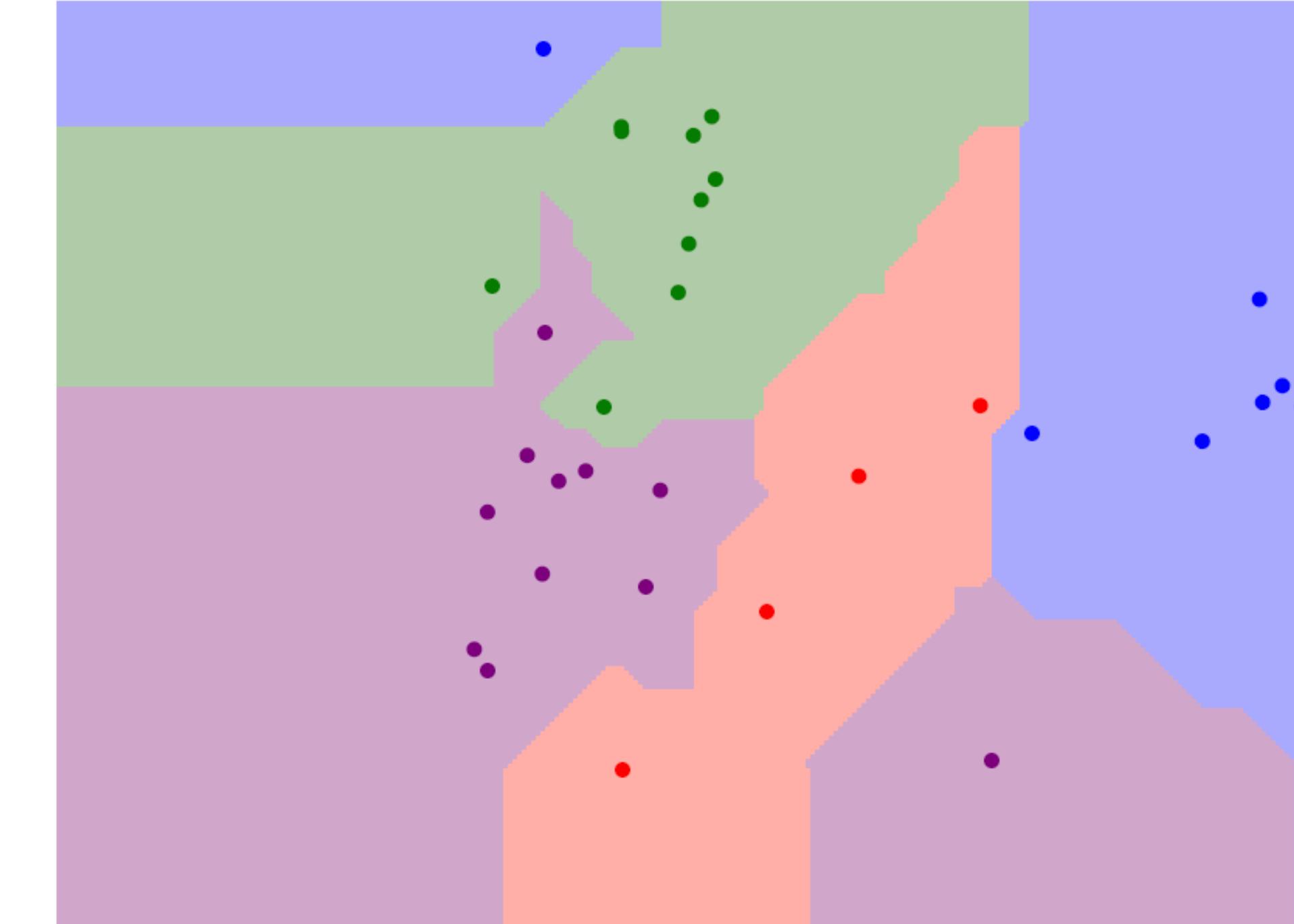
- **365 classes**
- **256x256**
- **8M/1.8M/.3M**
(training/validation/test)
- **Total number of non-empty synsets: 21841**
- **Total number of images: 14,197,122**
- **Number of images with bounding box annotations: 1,034,908**
- **Number of synsets with SIFT features: 1000**
- **Number of images with SIFT features: 1.2 million**



L2 versus L1 distance measure



L2 distance measure



L1 distance measure

L1 How to compare two images?

$$D(I_1, I_2) := \sum_{ij} |I_1(i, j) - I_2(i, j)|$$

L1 How to compare two images?

$$D(I_1, I_2) := \sum_{ij} |I_1(i, j) - I_2(i, j)|$$

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167

L1 How to compare two images?

$$D(I_1, I_2) := \sum_{ij} |I_1(i, j) - I_2(i, j)|$$

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167

37	8	14	200
20	140	70	35
15	1	195	100
9	99	55	170

L1 How to compare two images?

$$D(I_1, I_2) := \sum_{ij} |I_1(i, j) - I_2(i, j)|$$

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167

37	8	14	200
20	140	70	35
15	1	195	100
9	99	55	170

-

=

2	2	2	100
10	12	8	5
1	0	5	6
3	1	3	3

L1 How to compare two images?

$$D(I_1, I_2) := \sum_{ij} |I_1(i, j) - I_2(i, j)|$$

35	10	16	100
30	152	62	30
16	1	200	106
12	98	58	167

37	8	14	200
20	140	70	35
15	1	195	100
9	99	55	170

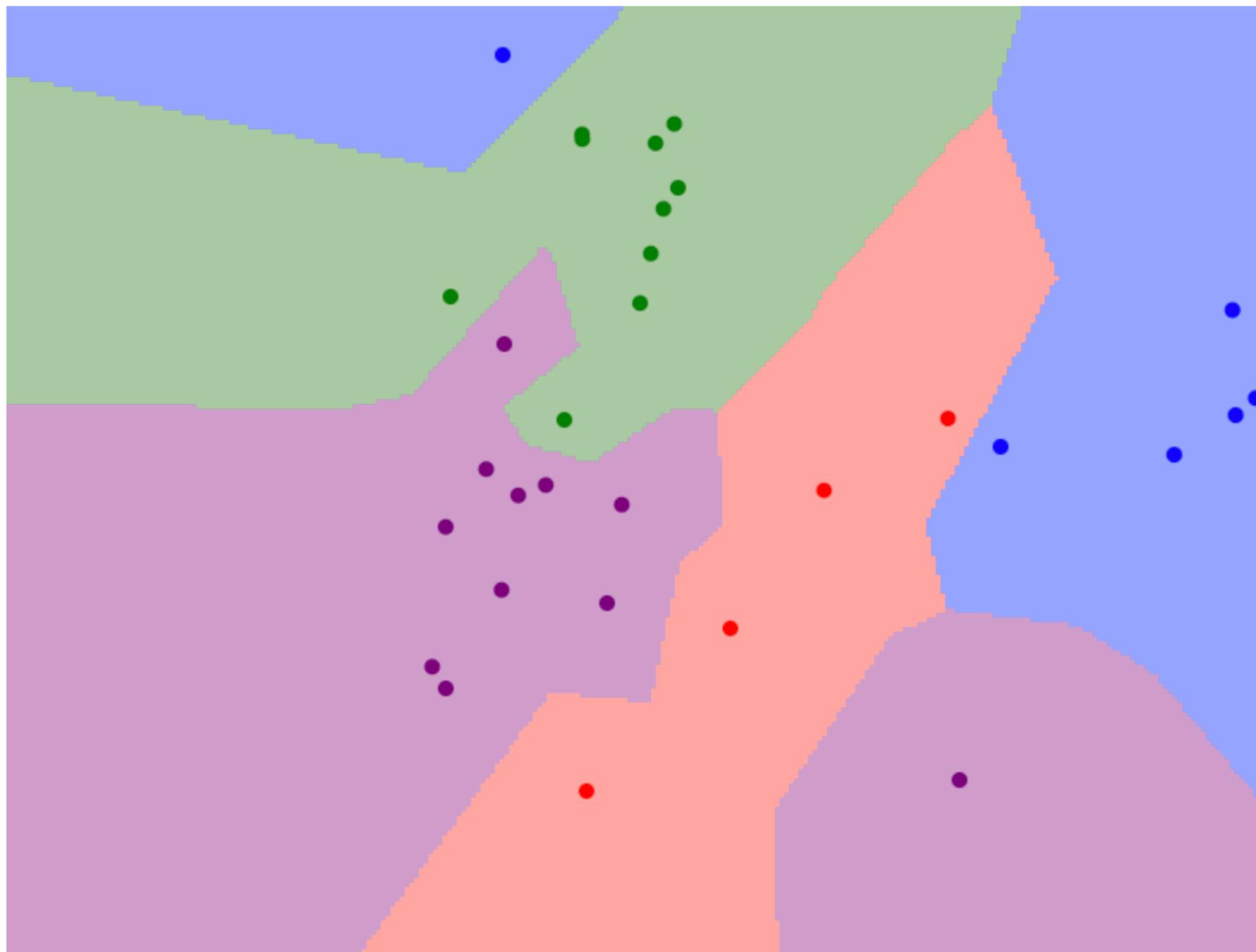
2	2	2	100
10	12	8	5
1	0	5	6
3	1	3	3

-

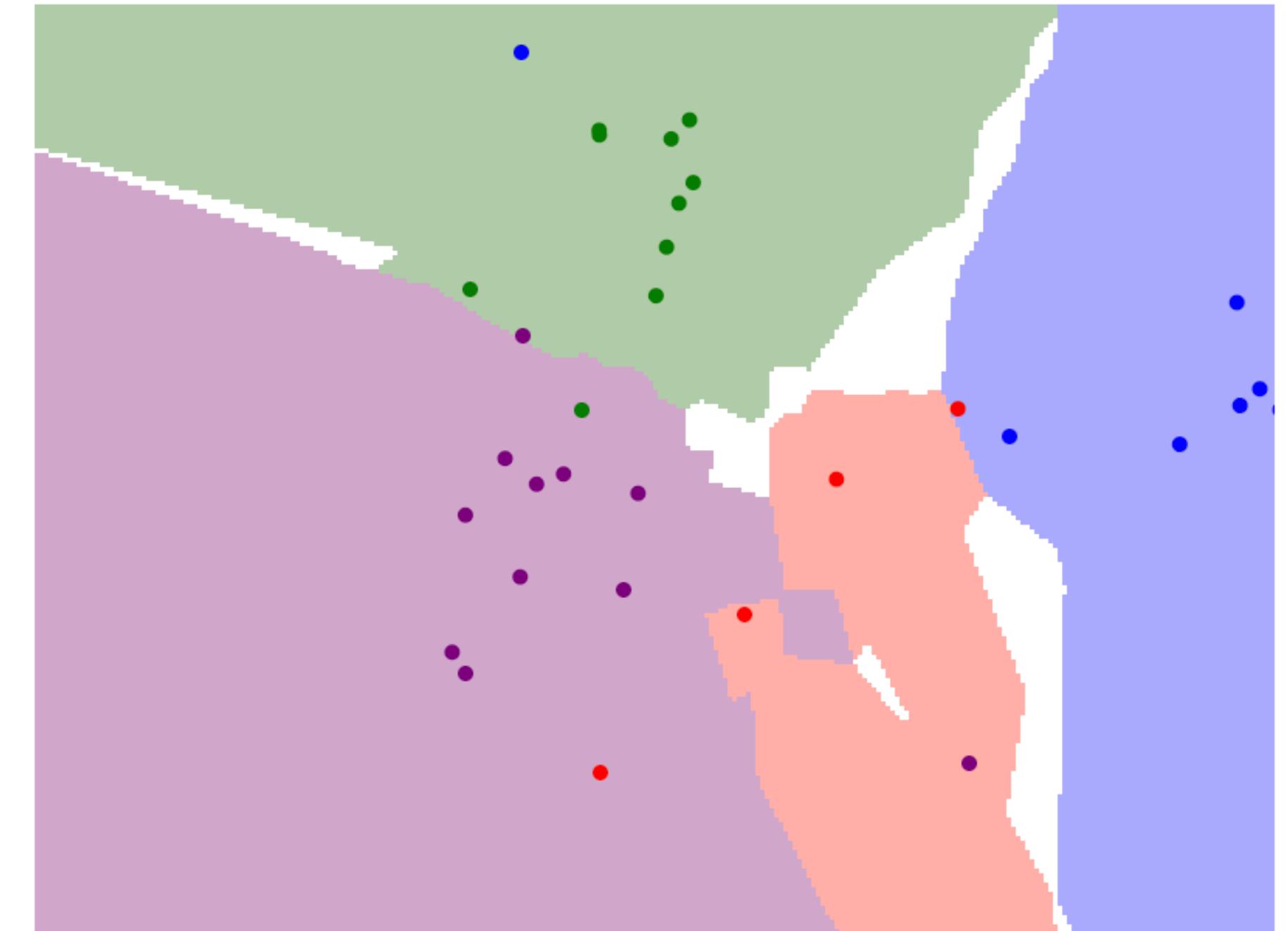
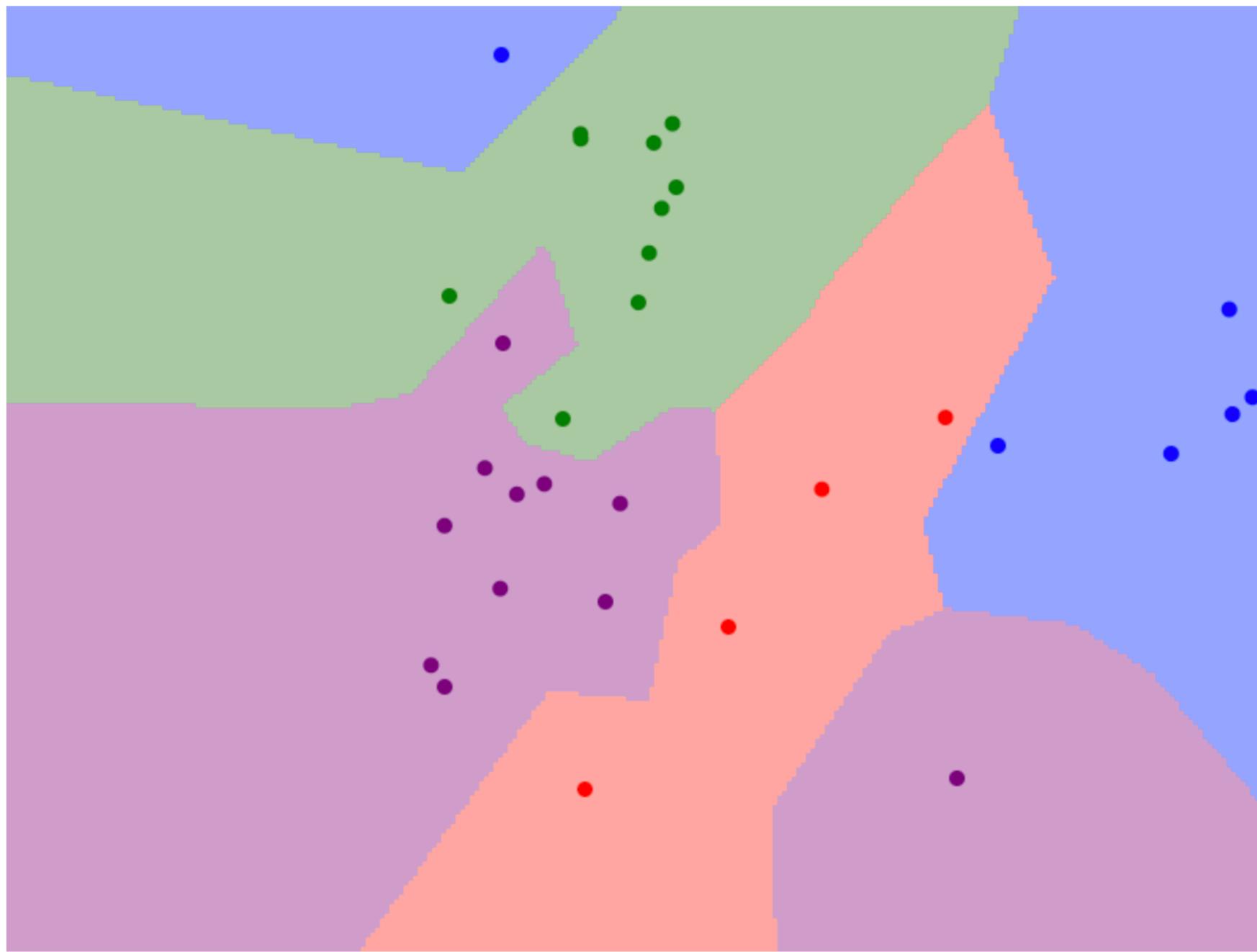
=

163

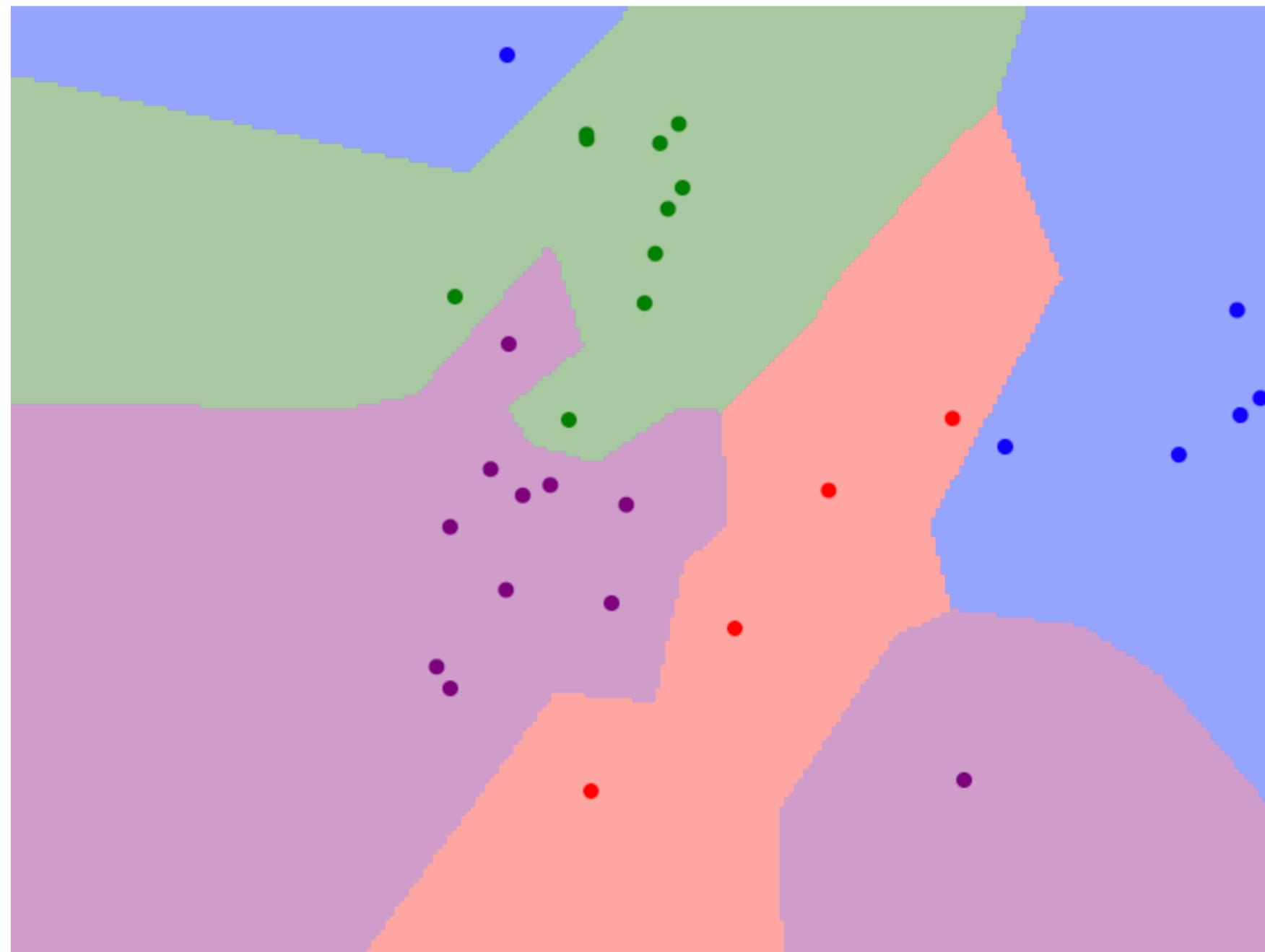
How to make this robust?



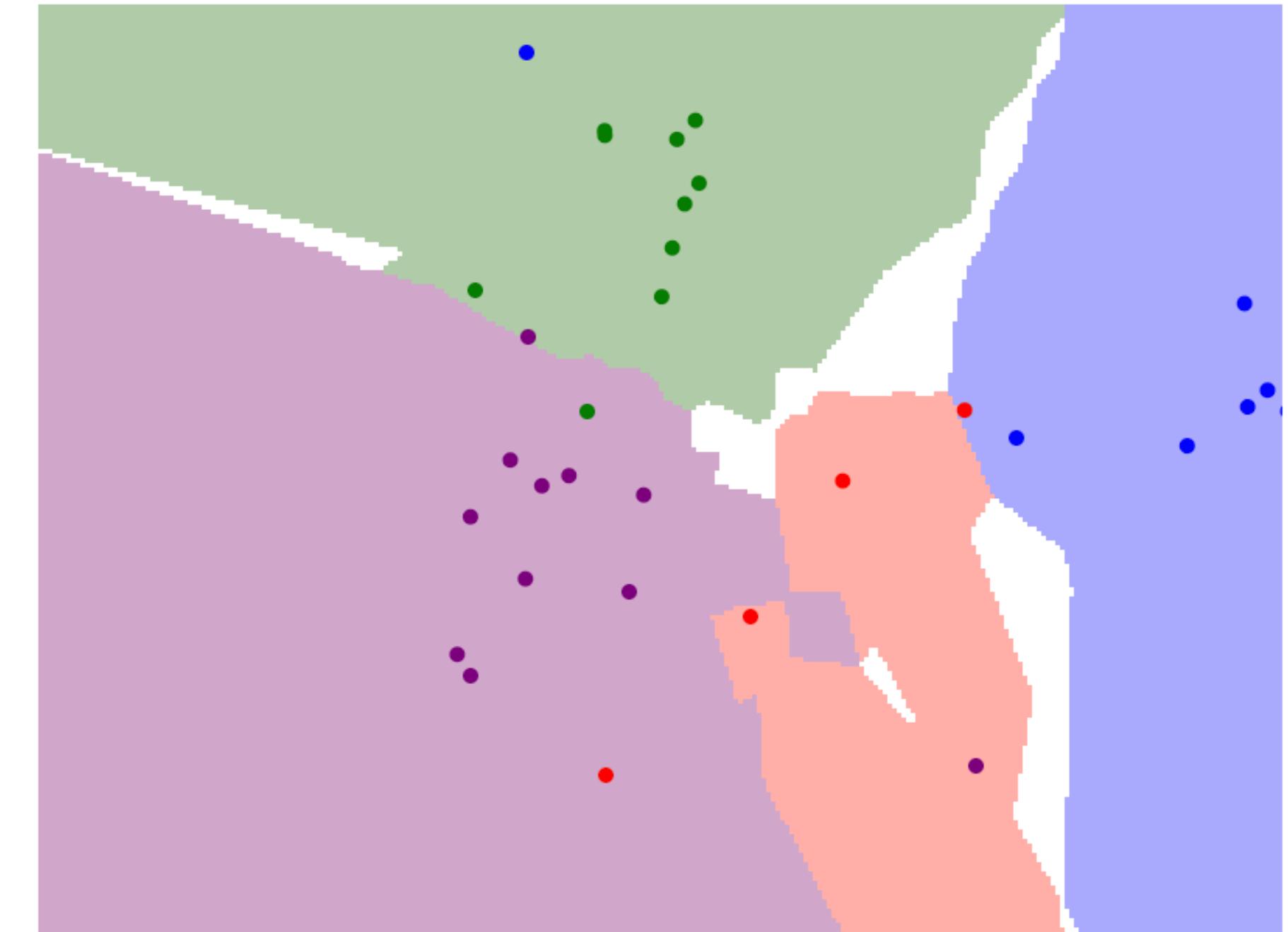
How to make this robust?



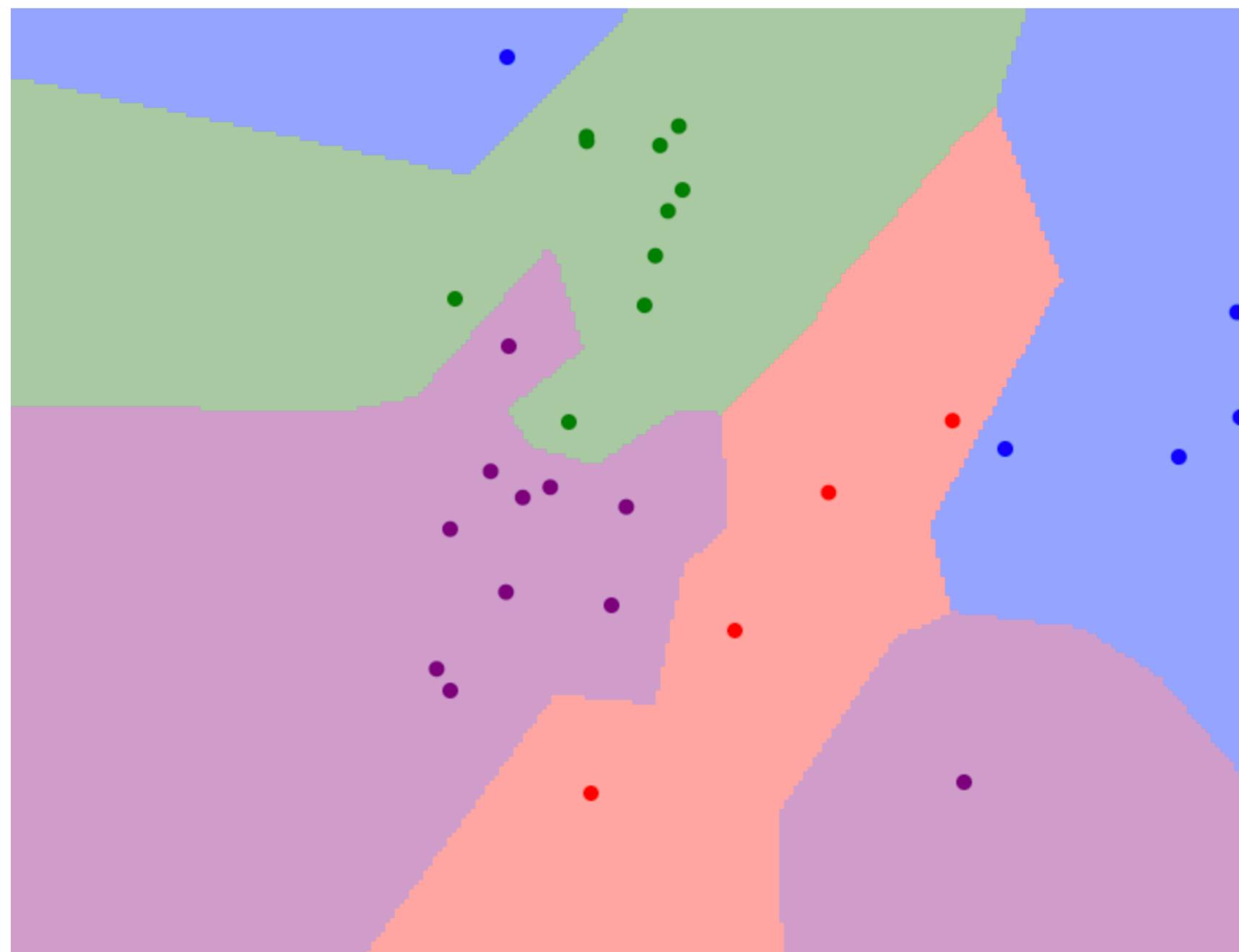
How to make this robust?



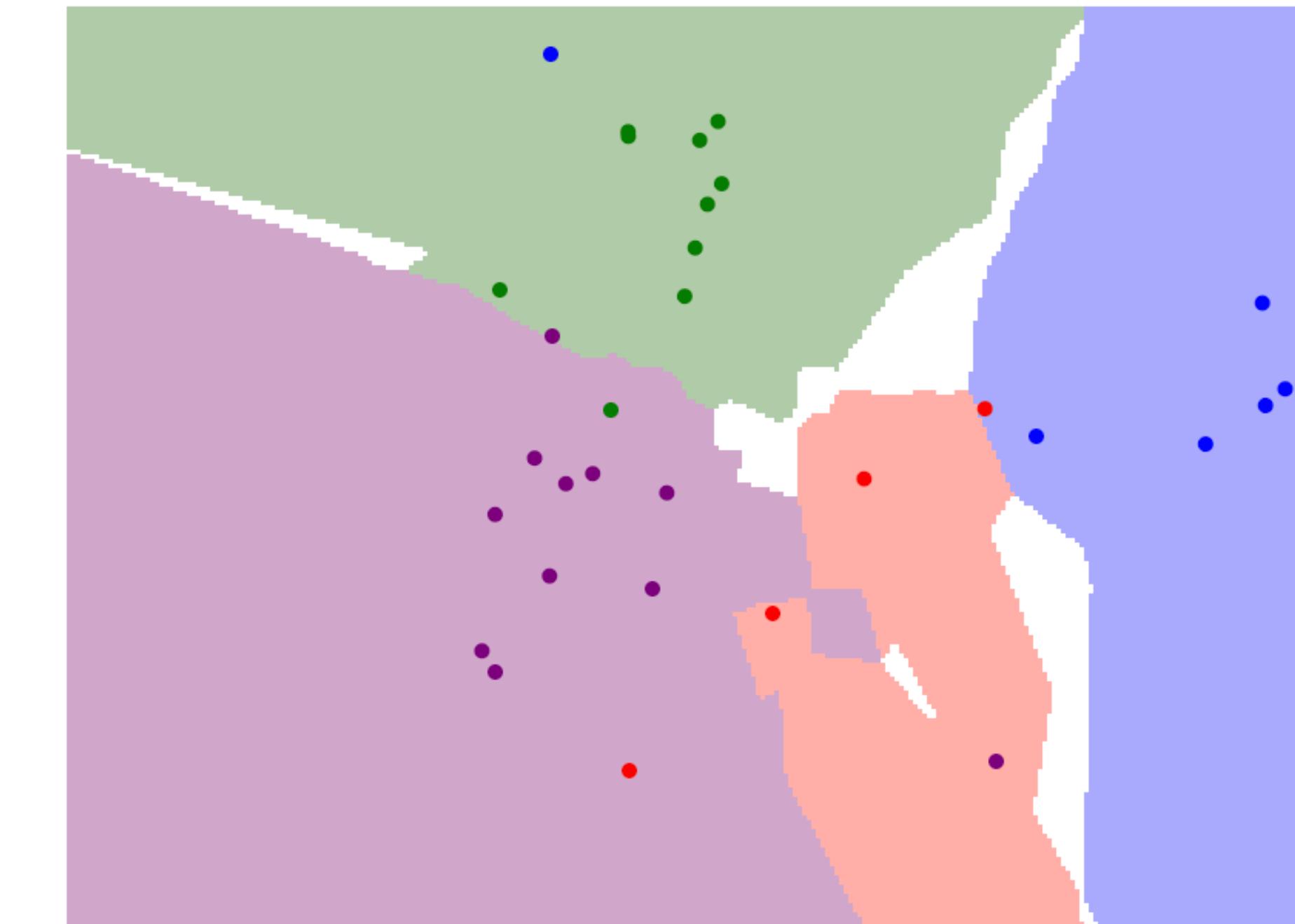
k=1



How to make this robust?



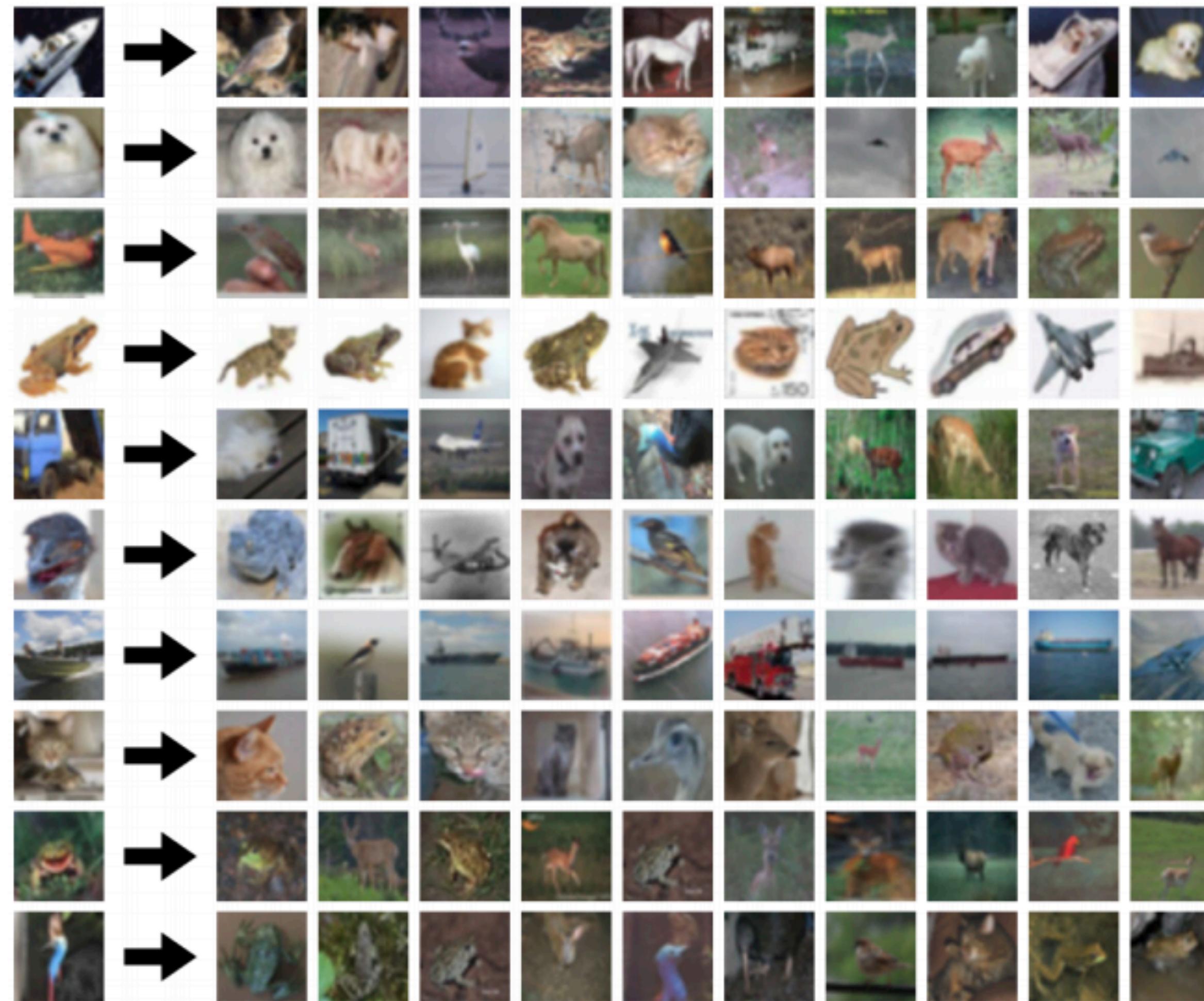
k=1



k=5

Take consensus among k neighbours.

How Does It Perform?



Hyperparameters

- **Choice of distance measure**
- **Choice of k — see multi-fold training later on**

kNN: Ambiguous on Raw Pixels

Original



Boxed



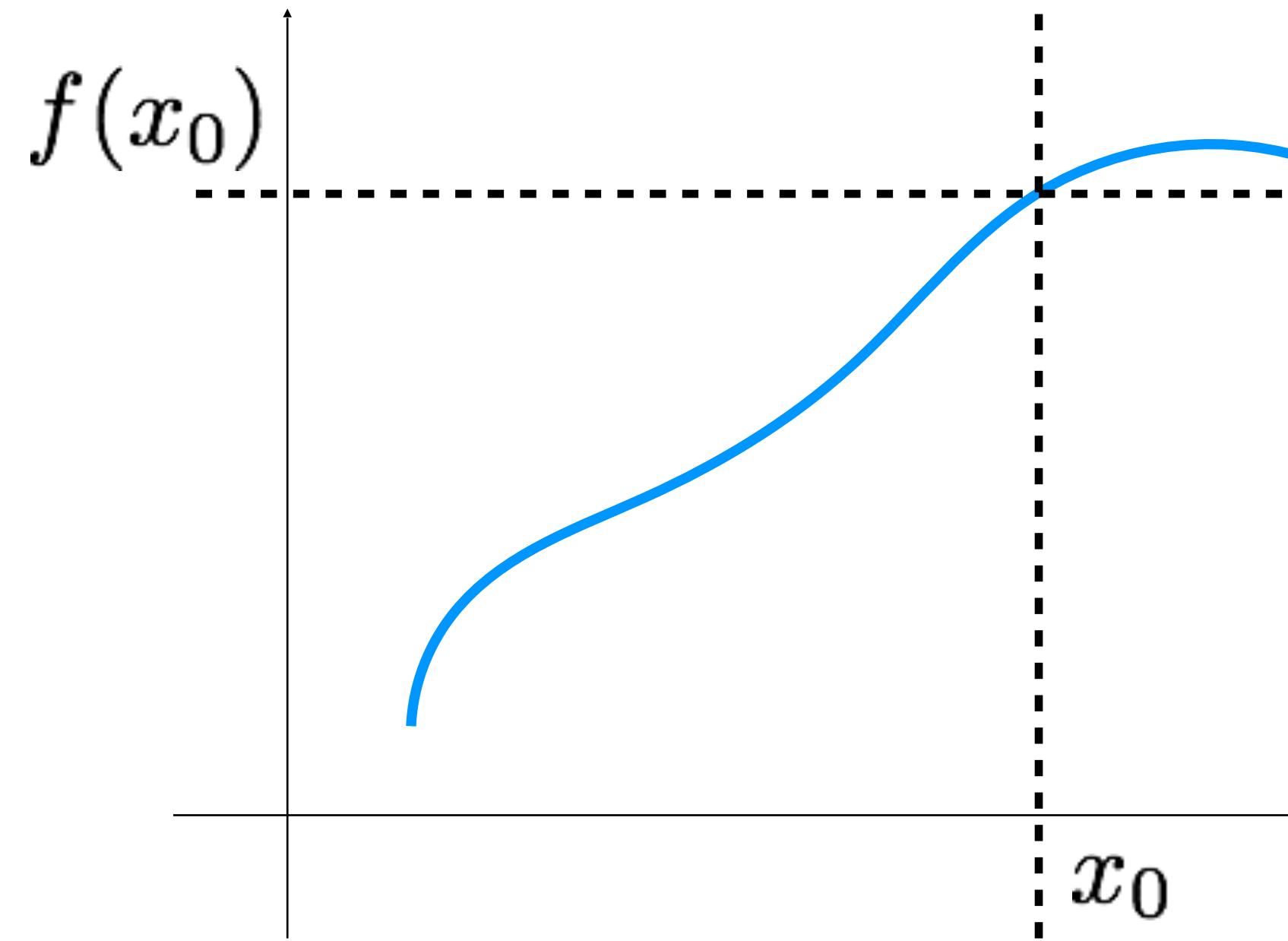
Shifted



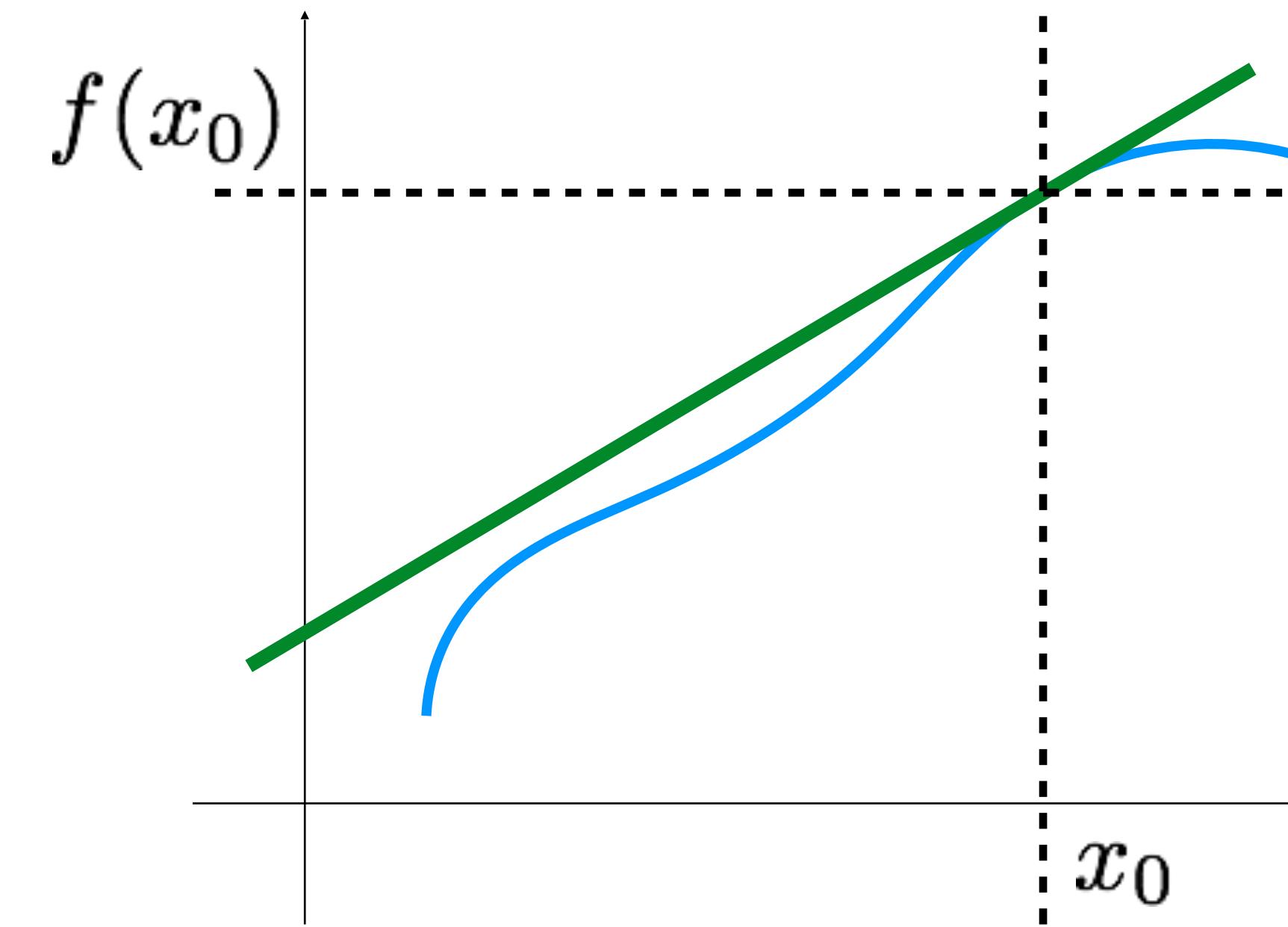
Tinted



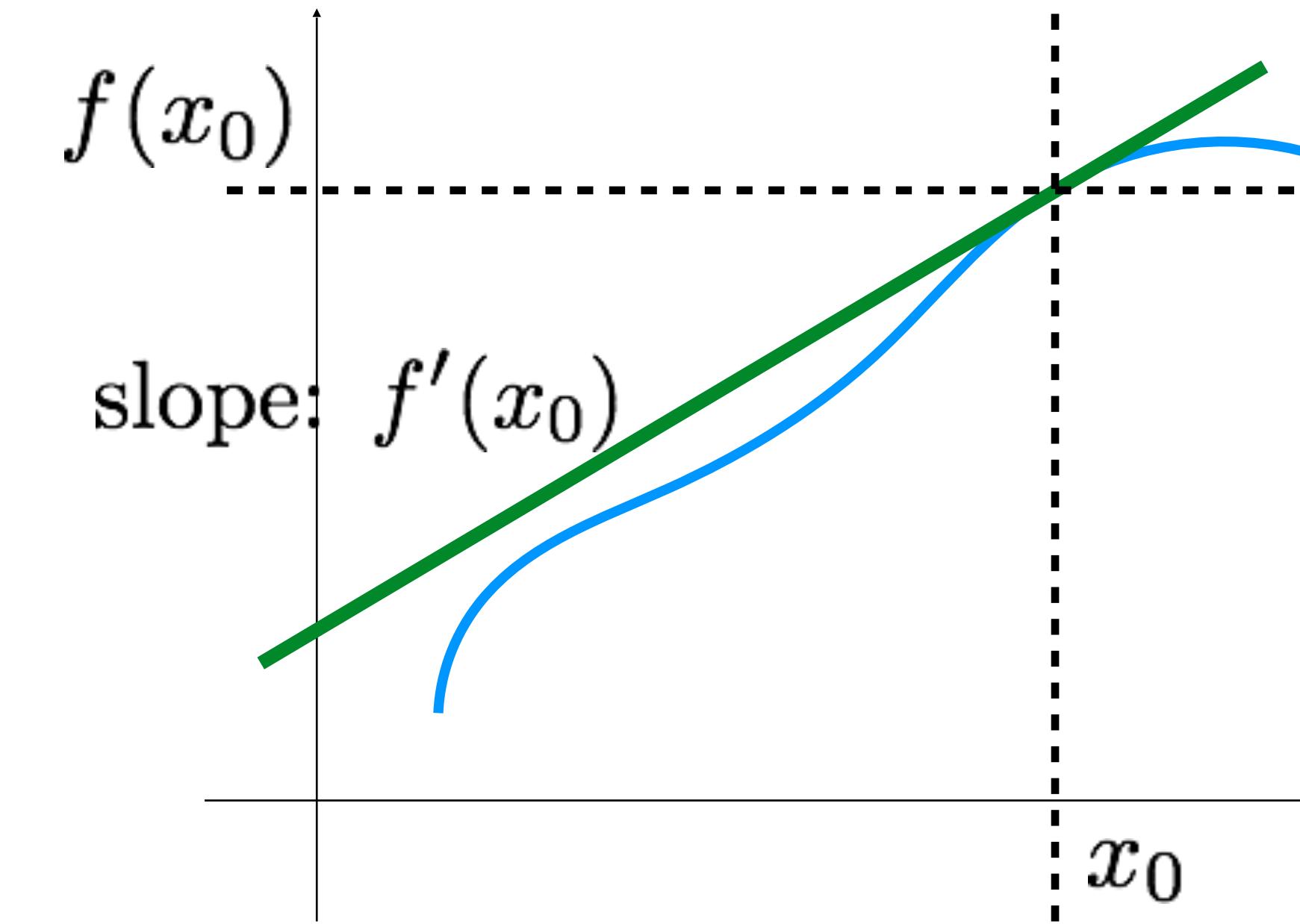
Differentiation (chain rule recap)



Differentiation (chain rule recap)

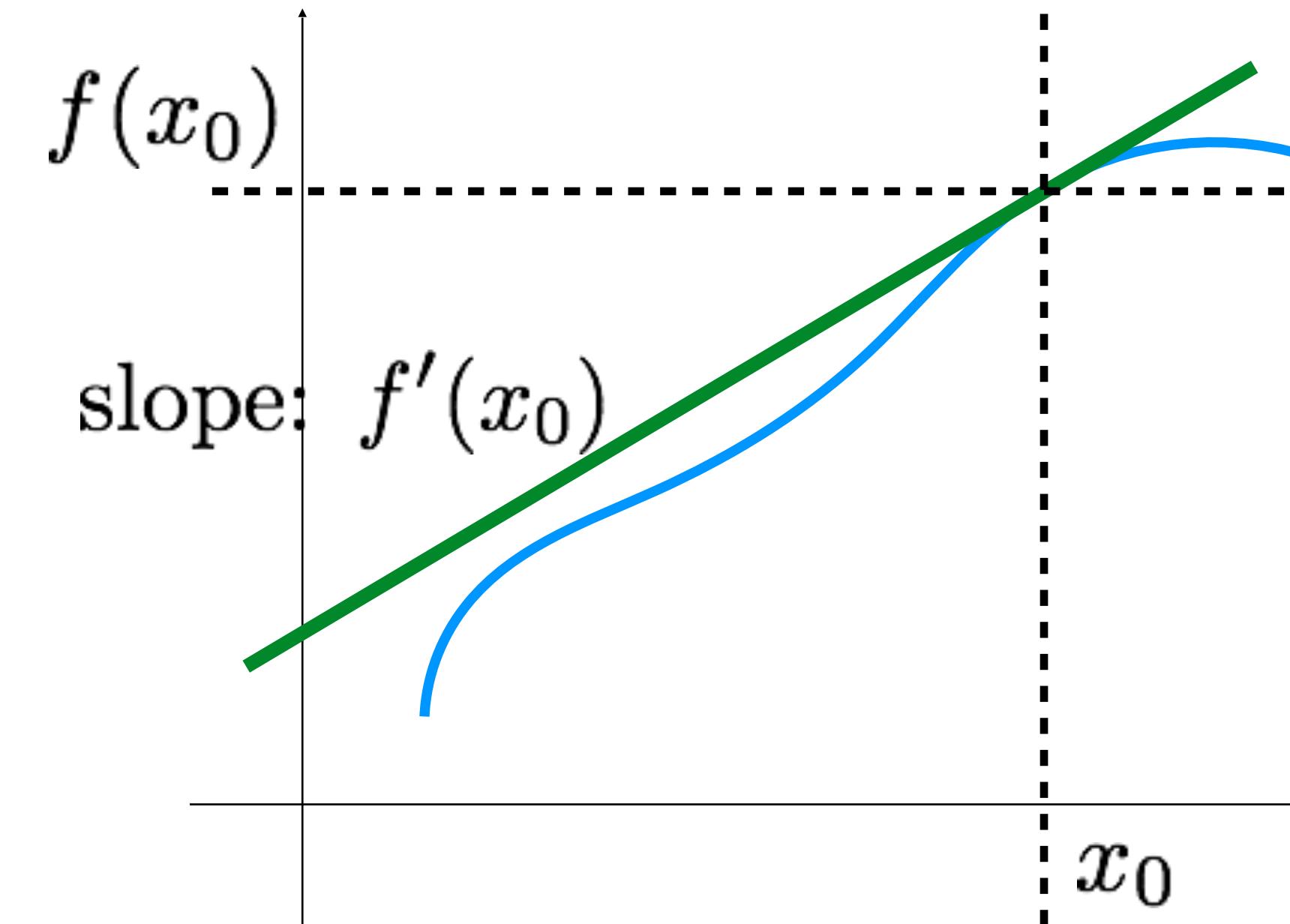


Differentiation (chain rule recap)



Differentiation (chain rule recap)

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

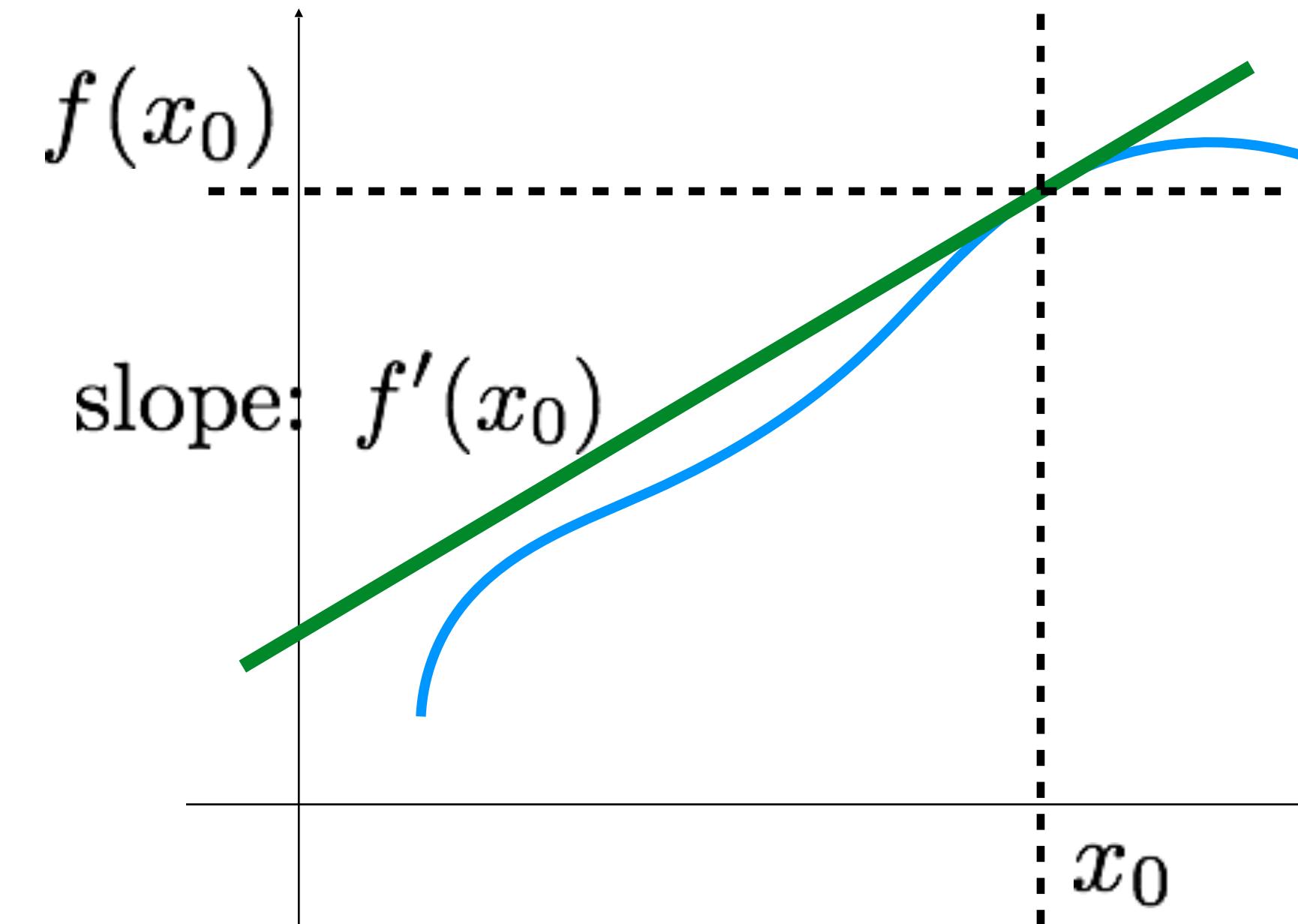


Differentiation (chain rule recap)

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

$$z = f(y)$$

$$y = g(x)$$



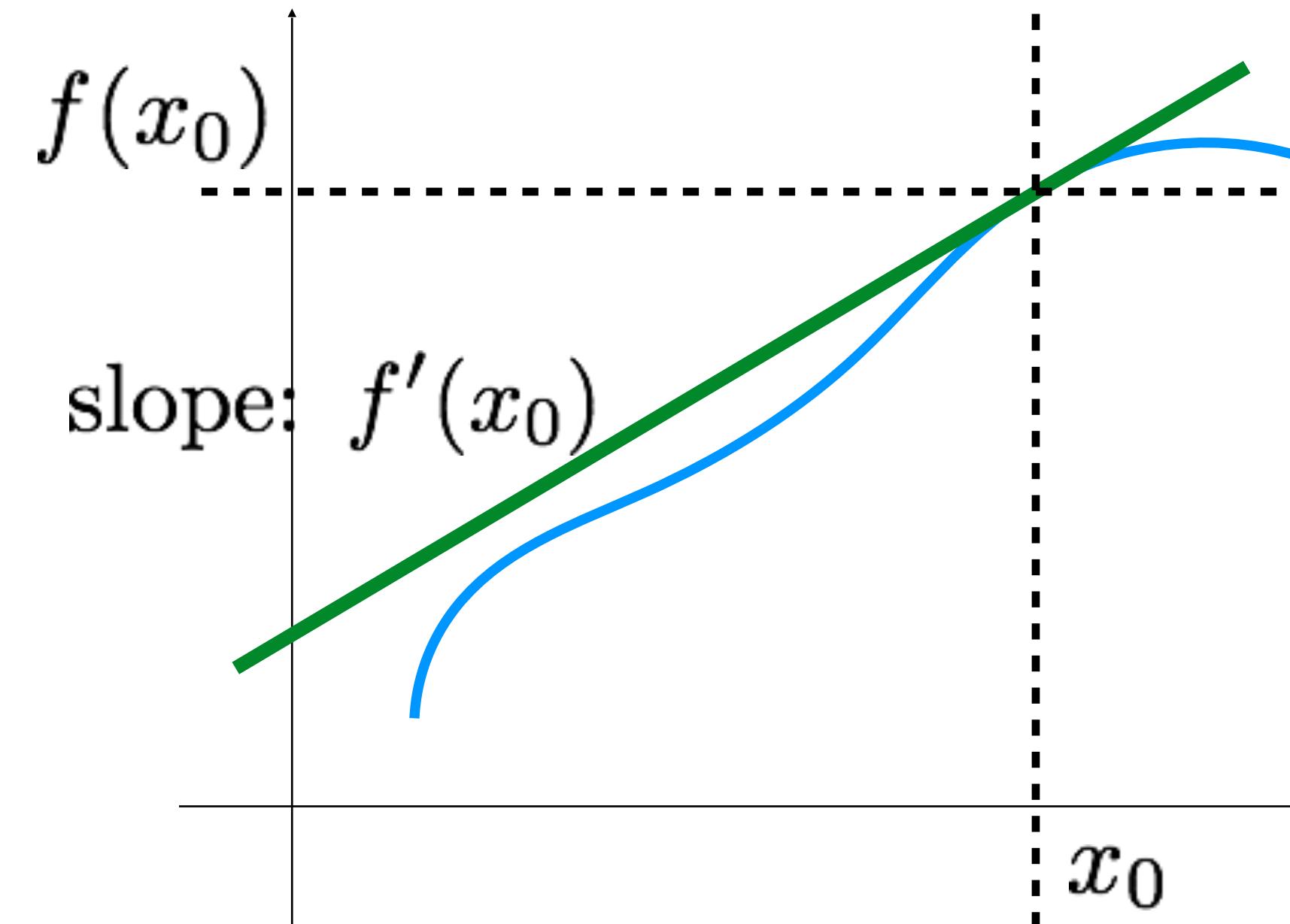
Differentiation (chain rule recap)

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

$$z = f(y)$$

$$y = g(x)$$

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



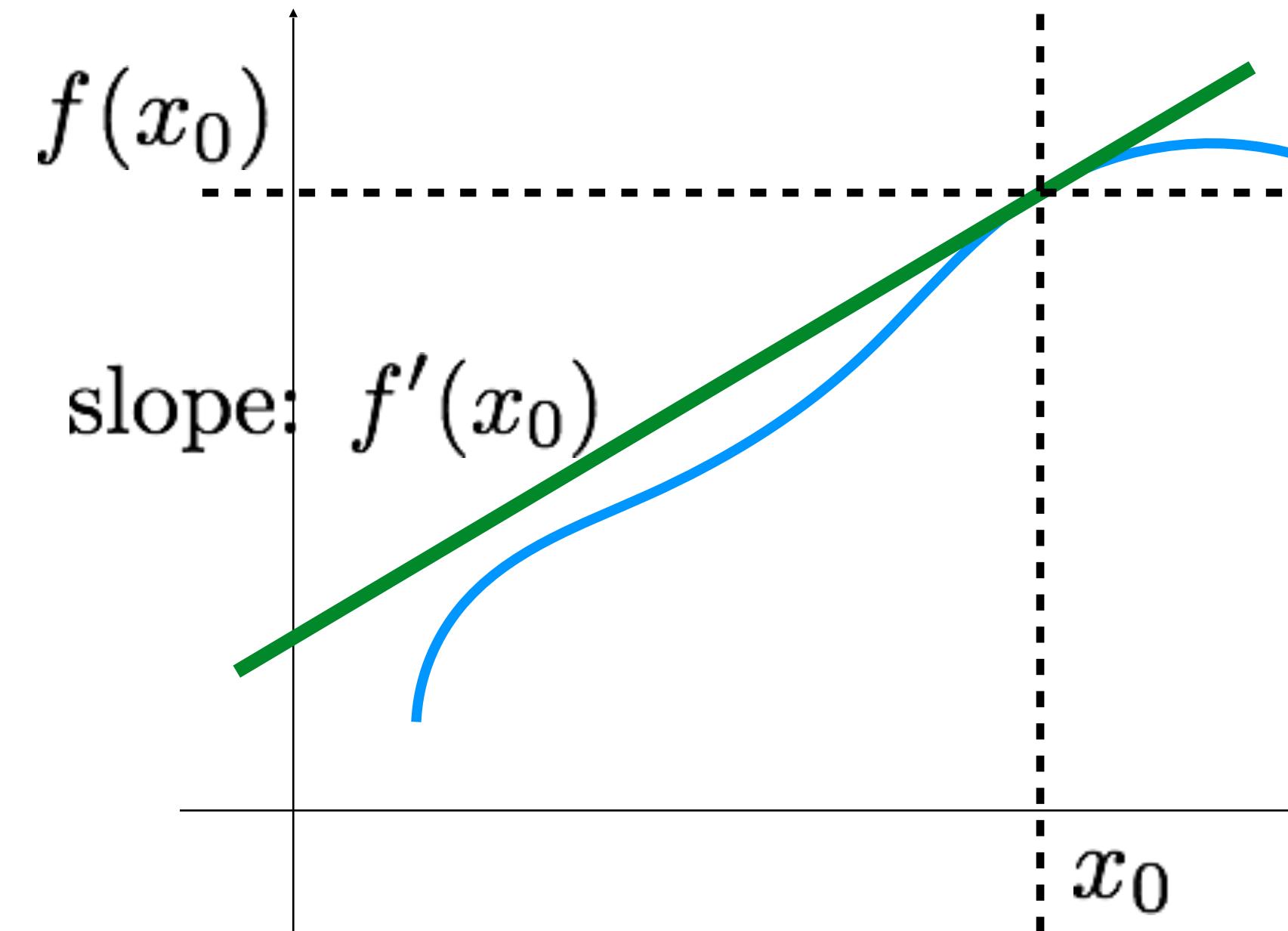
Differentiation (chain rule recap)

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

$$z = f(y)$$

$$y = g(x)$$

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



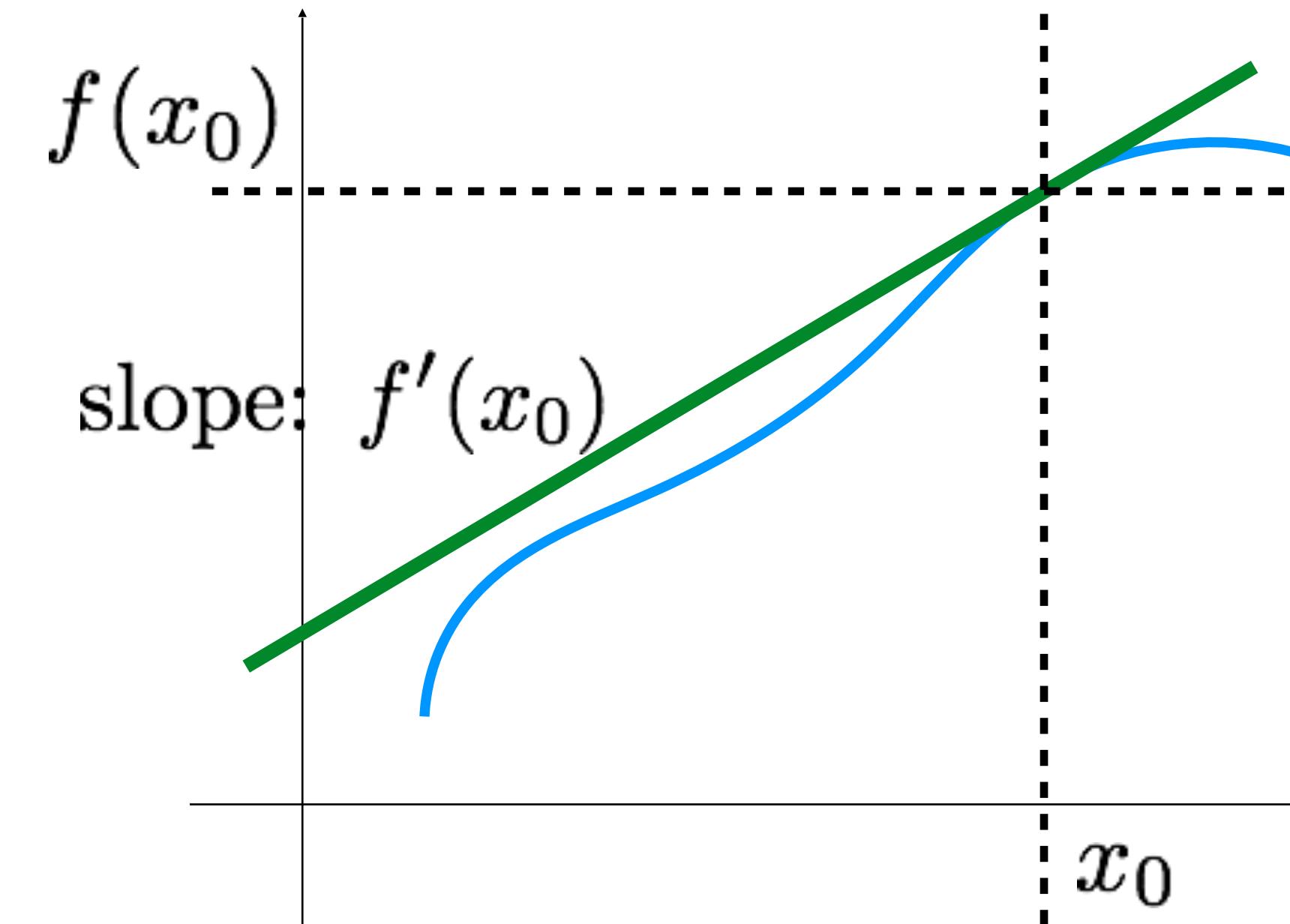
Differentiation (chain rule recap)

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

$$z = f(y)$$

$$y = g(x)$$

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



Differentiation (chain rule recap)

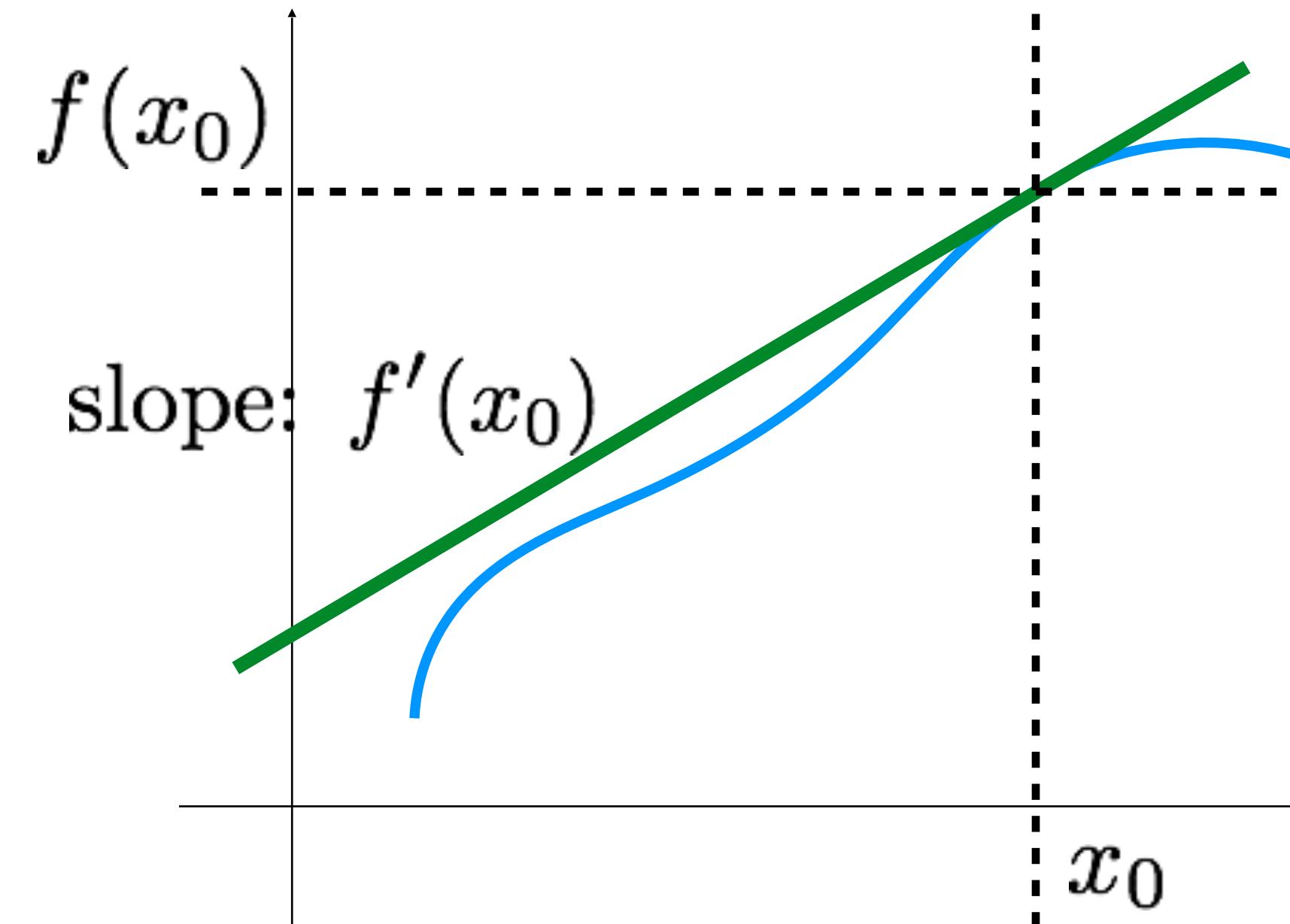
$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$

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

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



Differentiation (chain rule recap)

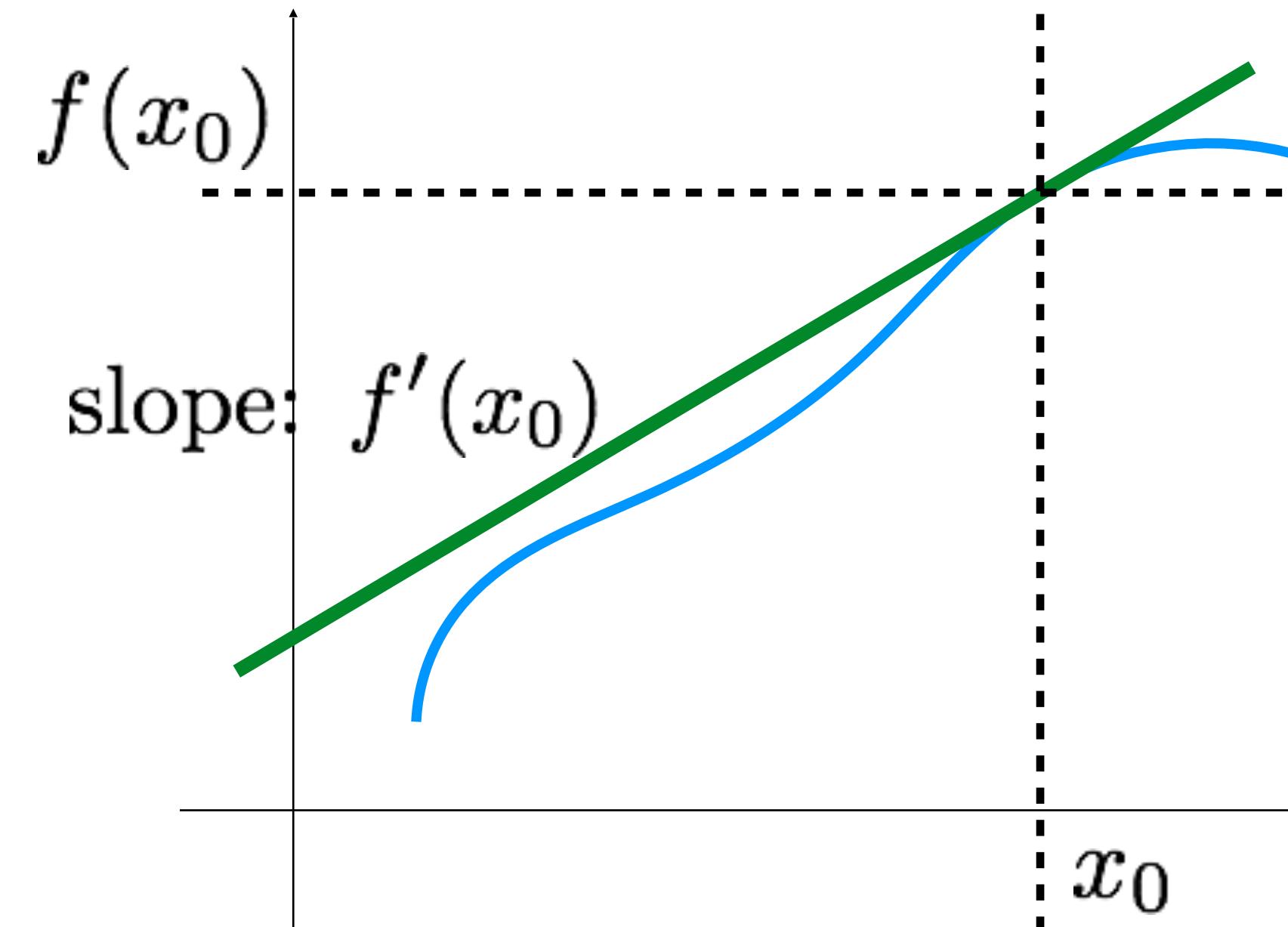
$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$

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

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



Differentiation (chain rule recap)

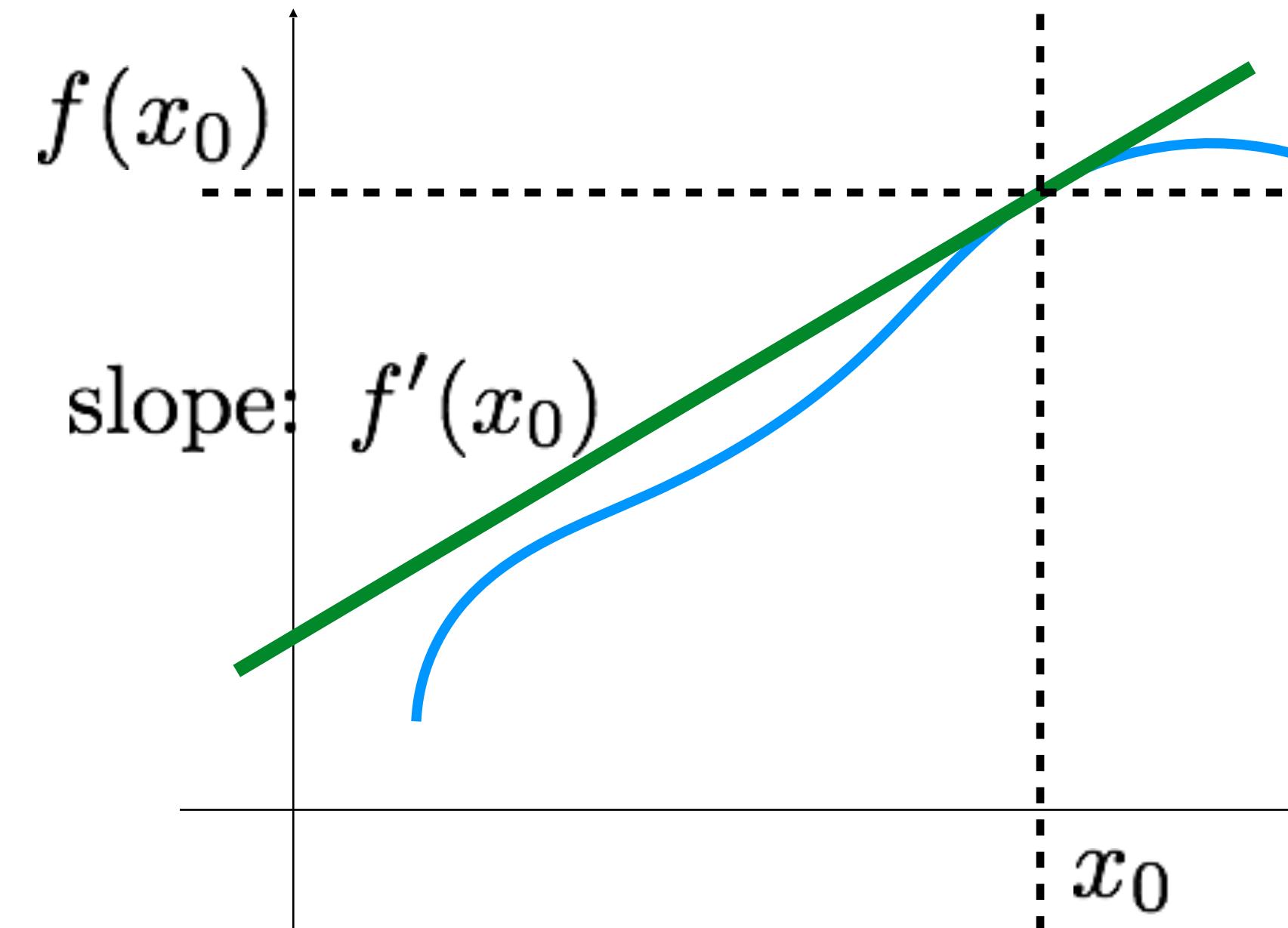
$$z = f \circ g(x) = f(g(x))$$

$$z = f(y)$$

$$y = g(x)$$

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

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



Derivative Matrix

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

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

Derivative Matrix

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

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

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

Derivative Matrix

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

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

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

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

Derivative Matrix

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

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

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

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

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

Derivative Matrix

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

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

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

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

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

Derivative Matrix

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

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

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

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

$$\mathbf{L} = D\mathbf{f} = \left[\frac{\partial f(\mathbf{x})}{\partial x_1} \quad \dots \quad \frac{\partial f(\mathbf{x})}{\partial x_n} \right]_{m \times n}$$

Derivative Matrix

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

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

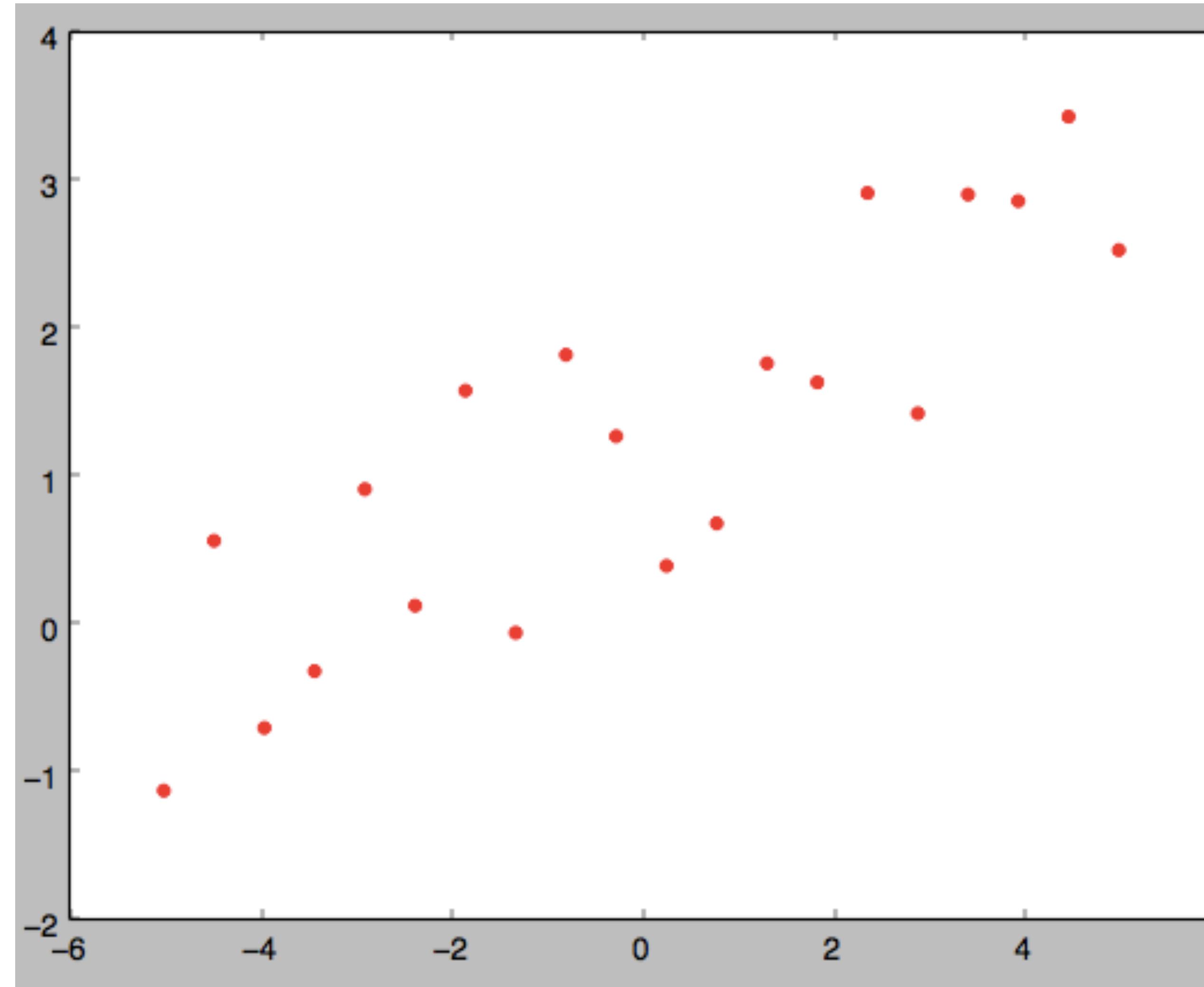
$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix}$$

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

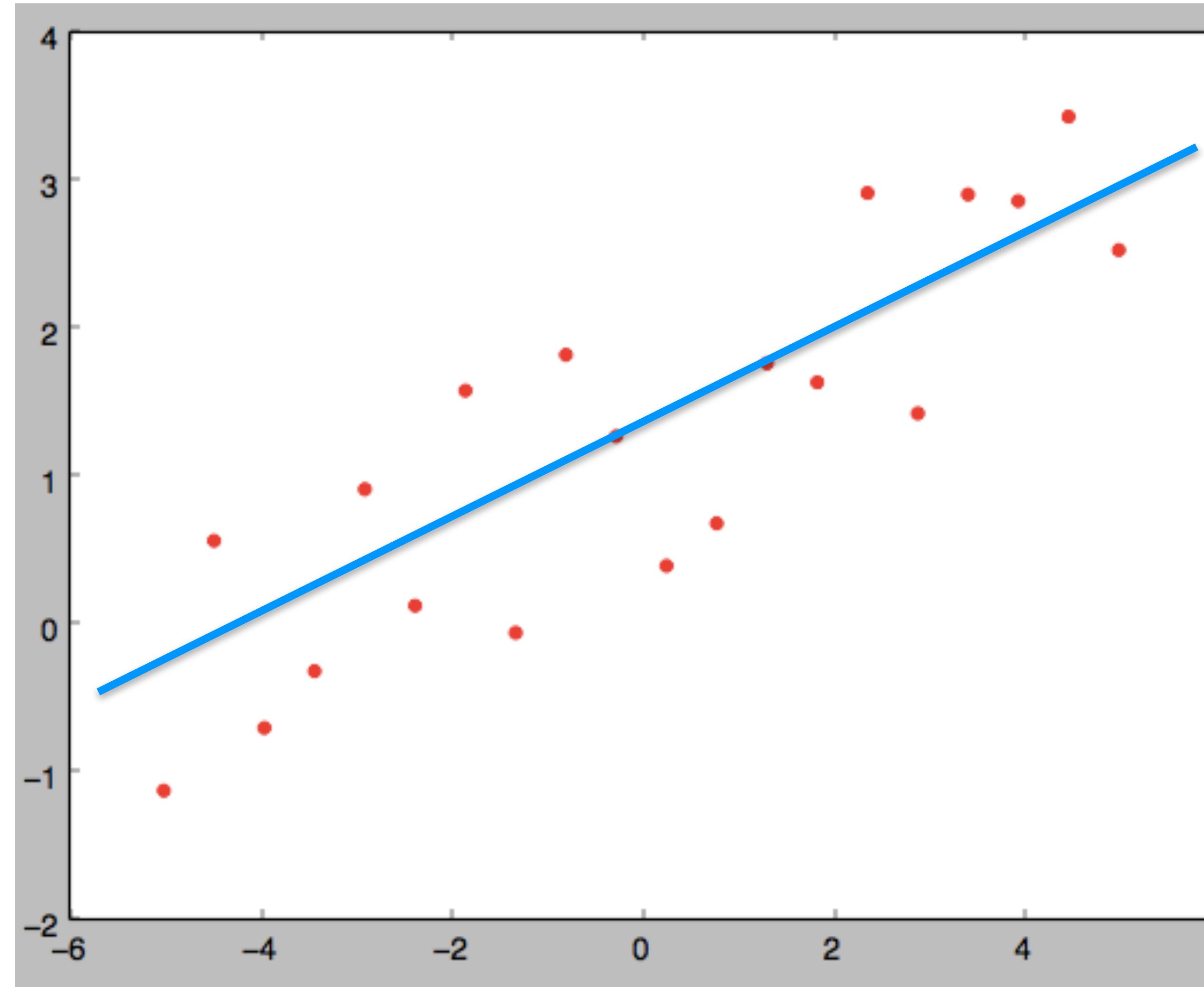
Jacobian matrix

$$\mathbf{L} = D\mathbf{f} = \left[\frac{\partial f(\mathbf{x})}{\partial x_1} \quad \dots \quad \frac{\partial f(\mathbf{x})}{\partial x_n} \right]_{m \times n}$$

Regression: Continuous Output



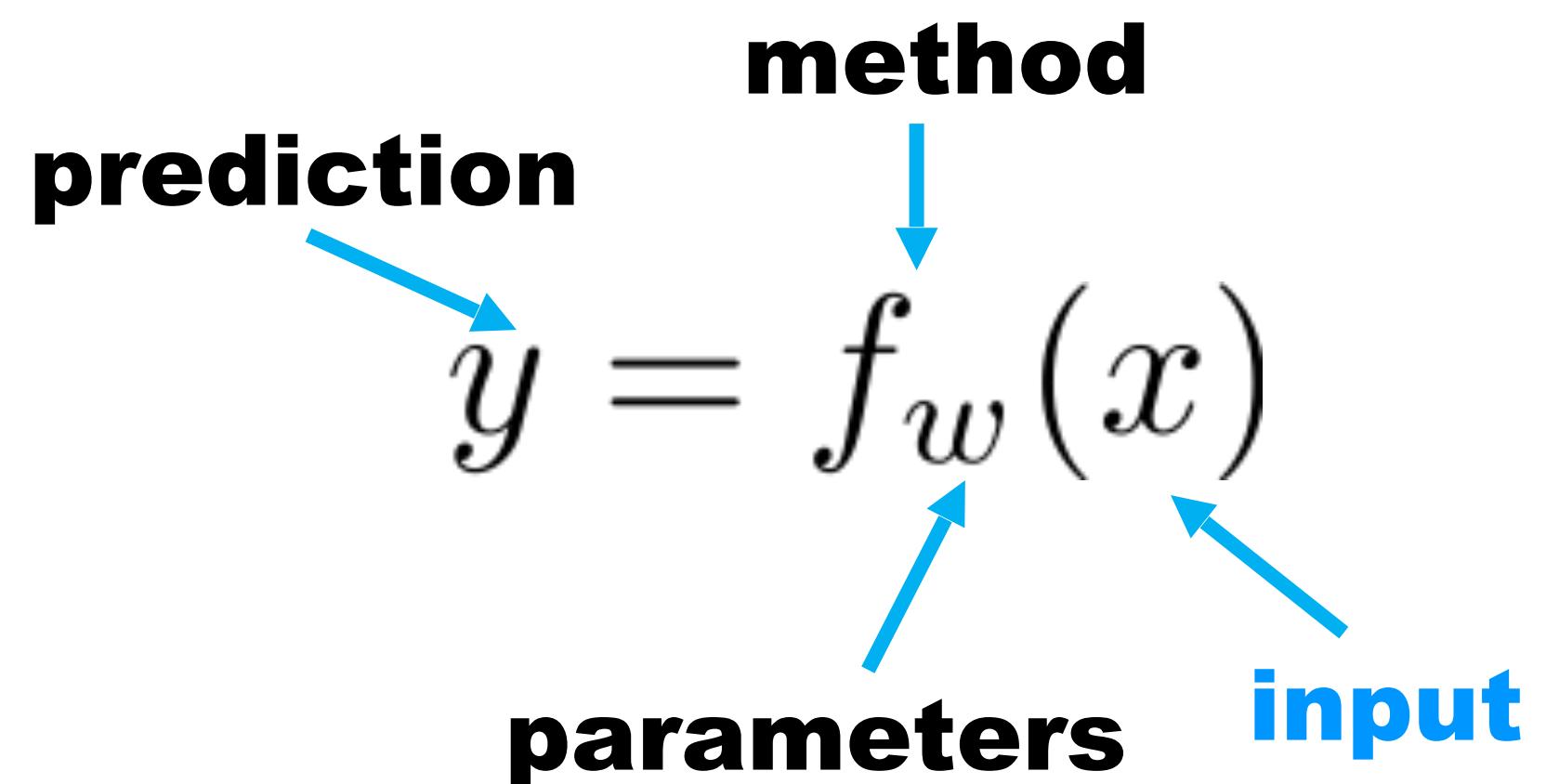
Regression: Continuous Output



Learning a Function

$$y = f_w(x)$$

Learning a Function



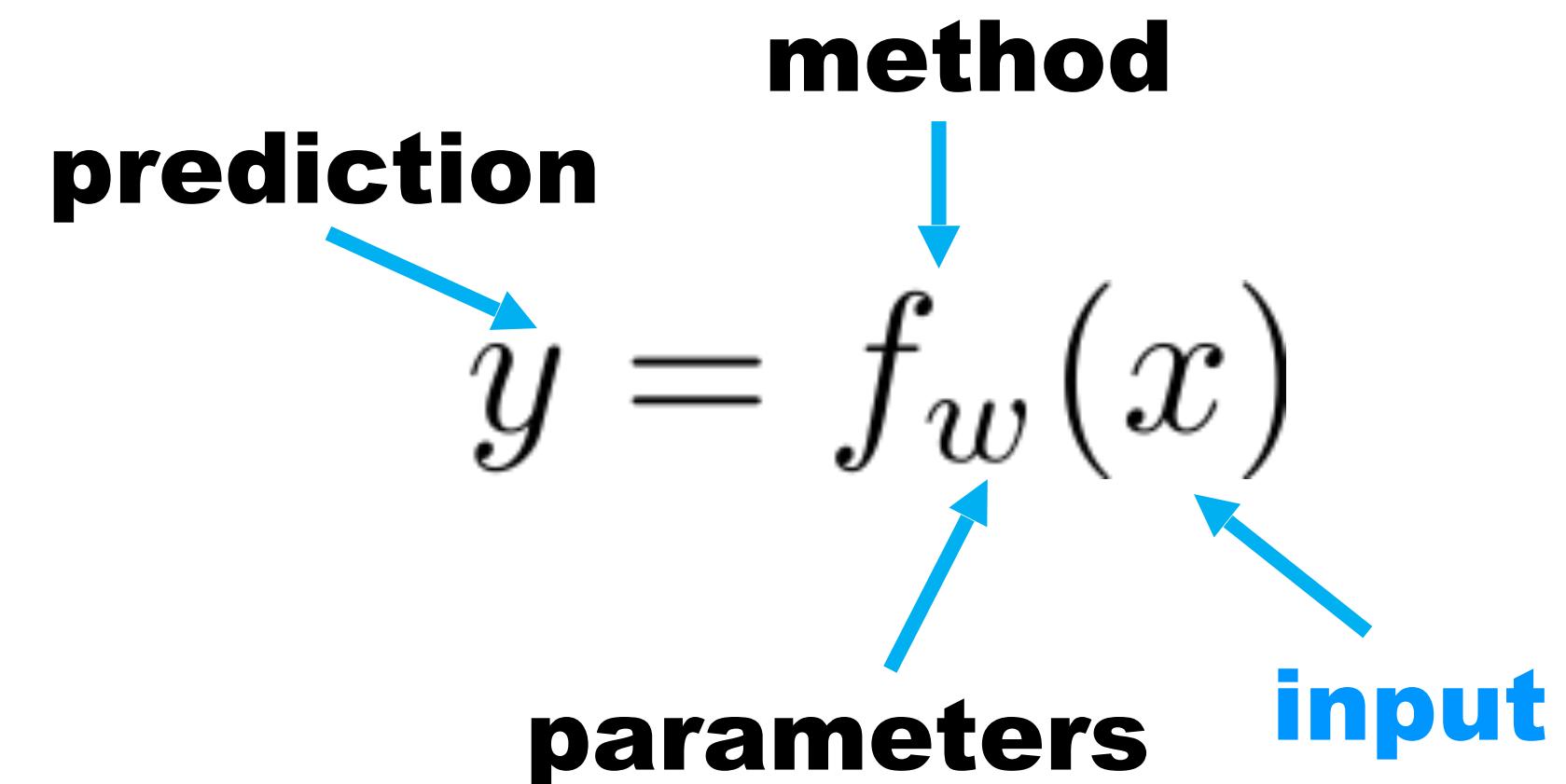
Calculus

$$x \in \mathbb{R}$$

Vector calculus

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

Learning a Function



Calculus

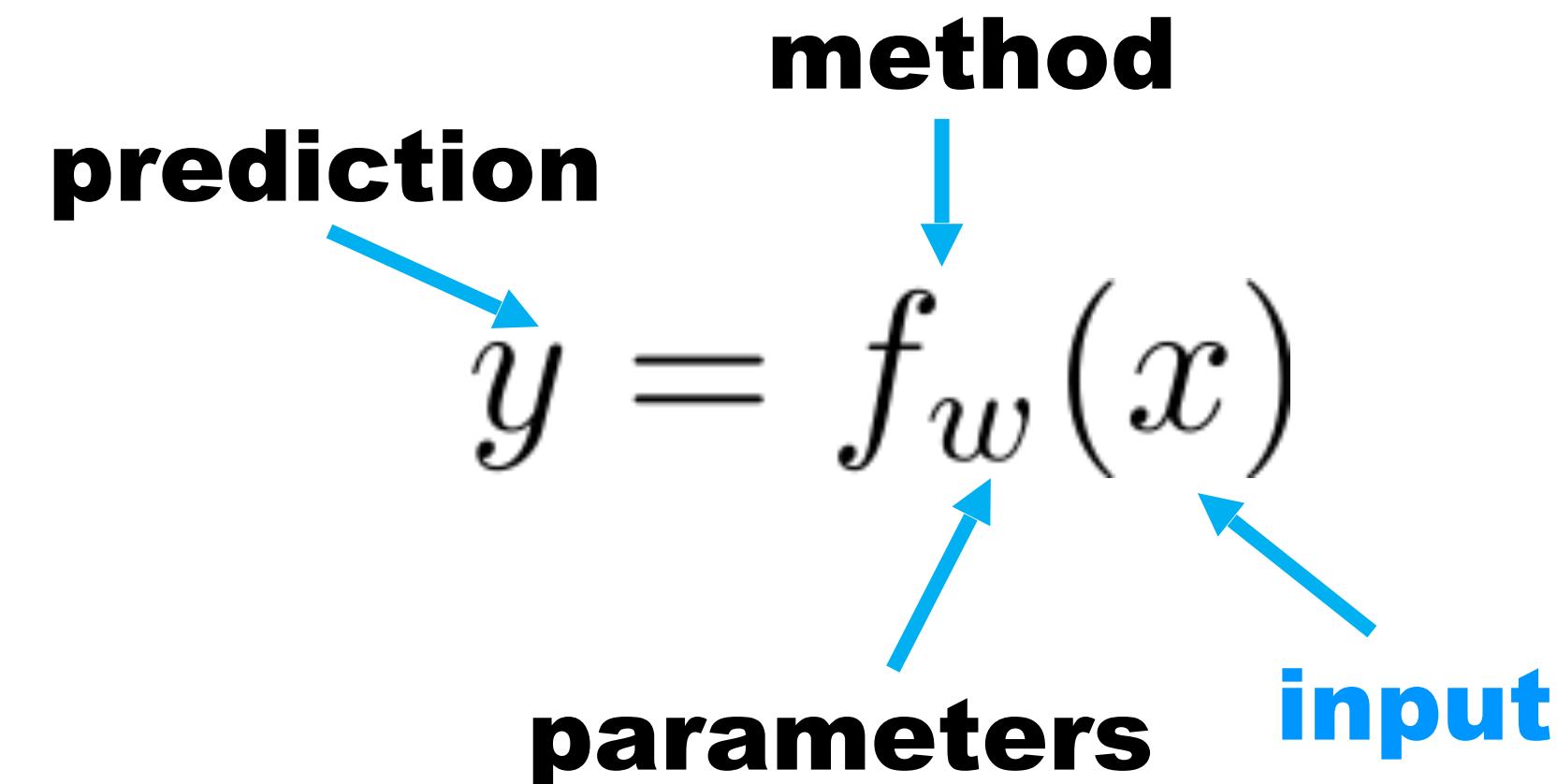
$$x \in \mathbb{R}$$

Vector calculus

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

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

Learning a Function



Calculus

$$x \in \mathbb{R}$$

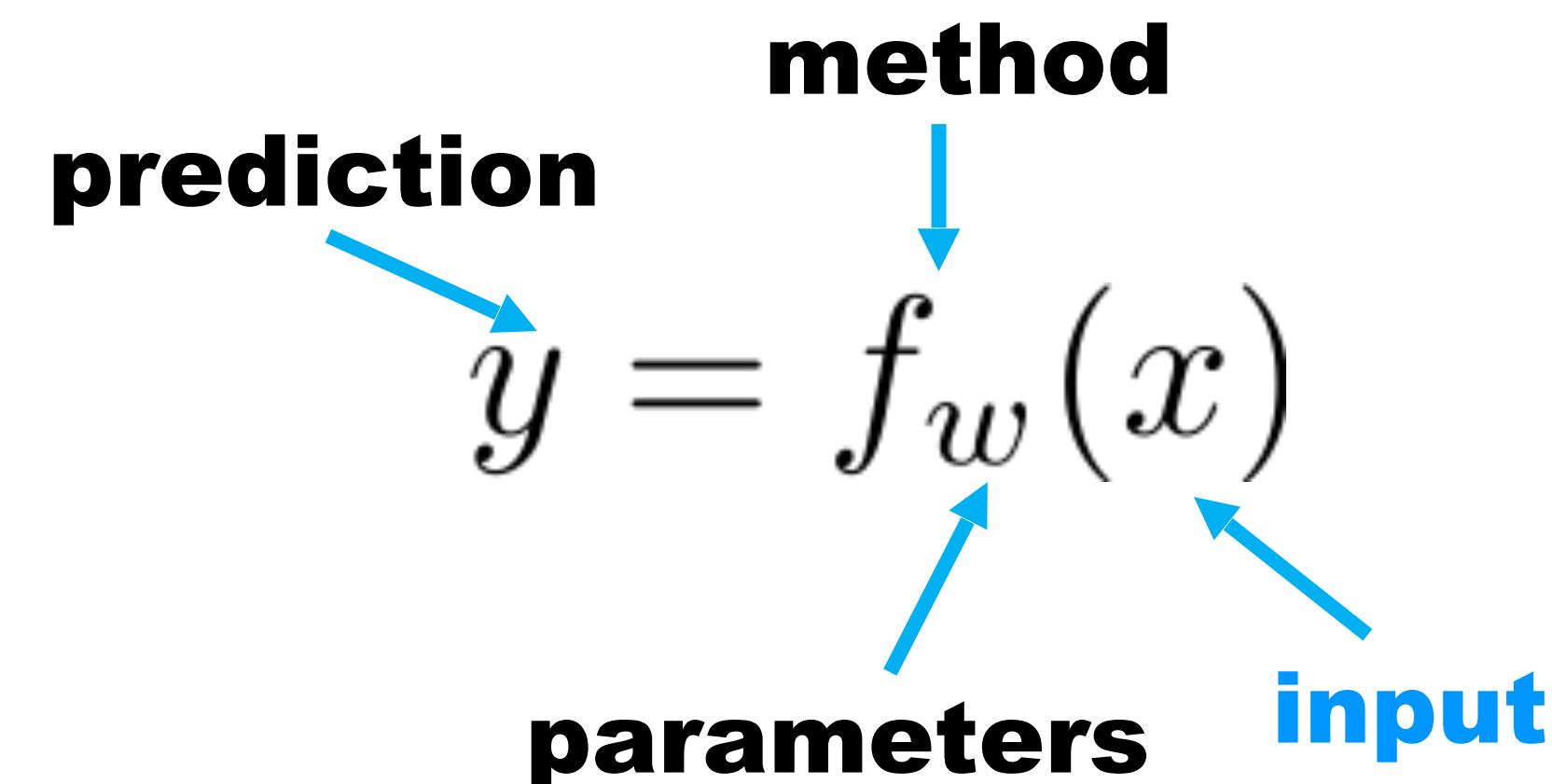
Classification: $y \in \{0, 1\}$

Vector calculus

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

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

Learning a Function



Calculus

$$x \in \mathbb{R}$$

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

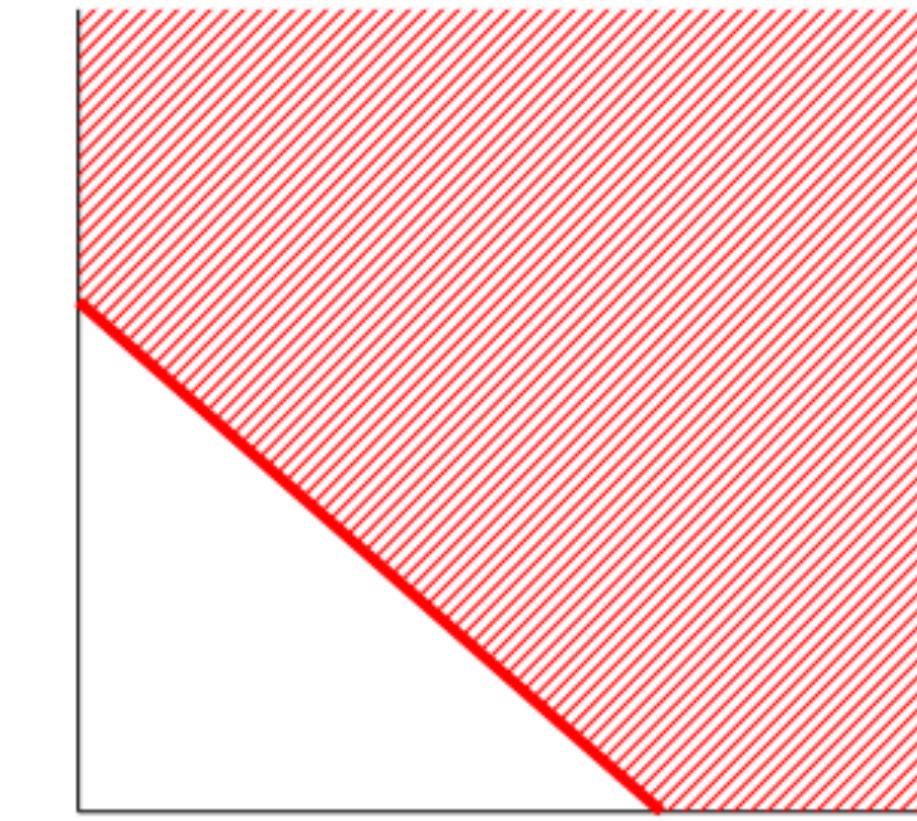
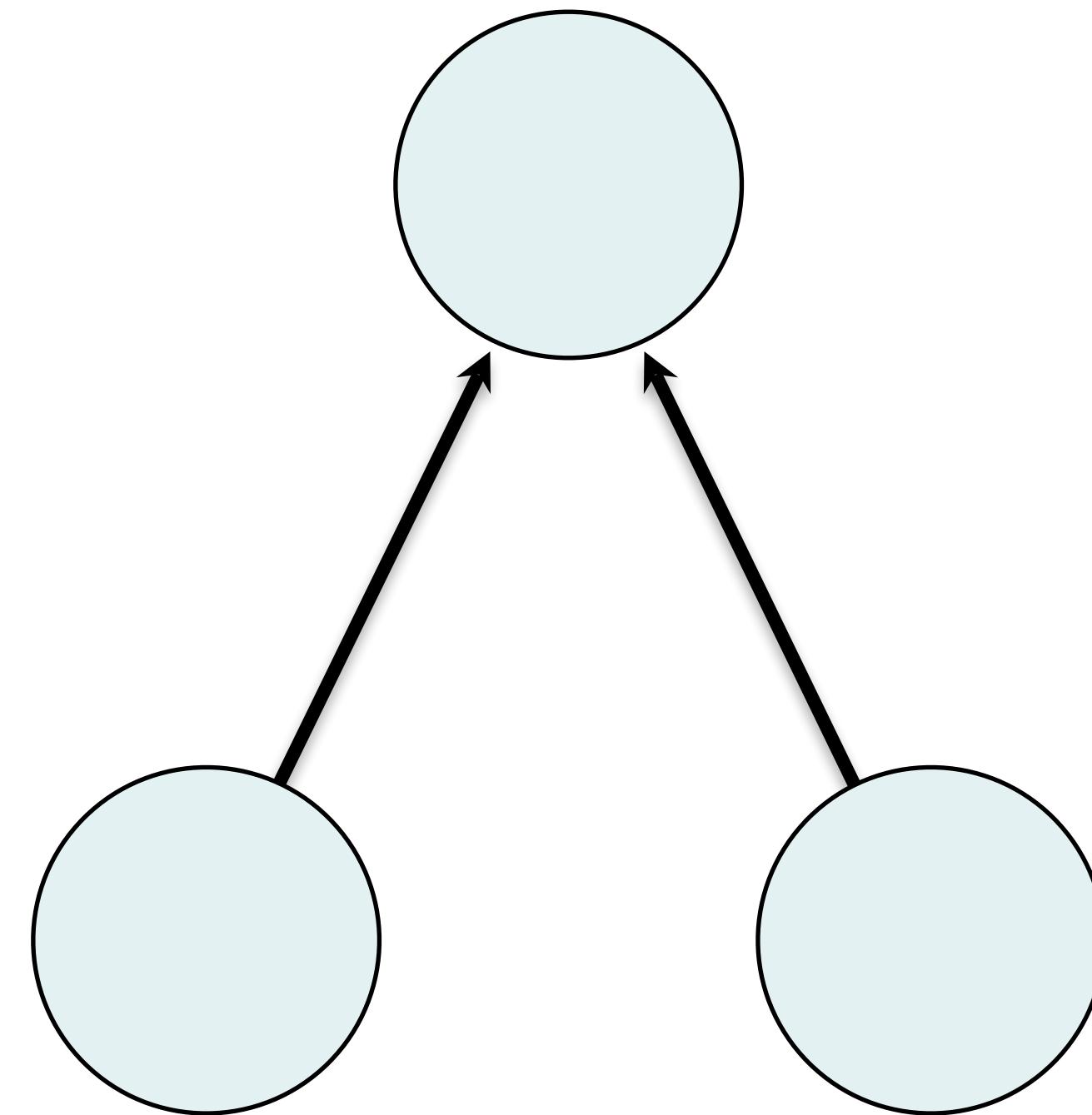
Vector calculus

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

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

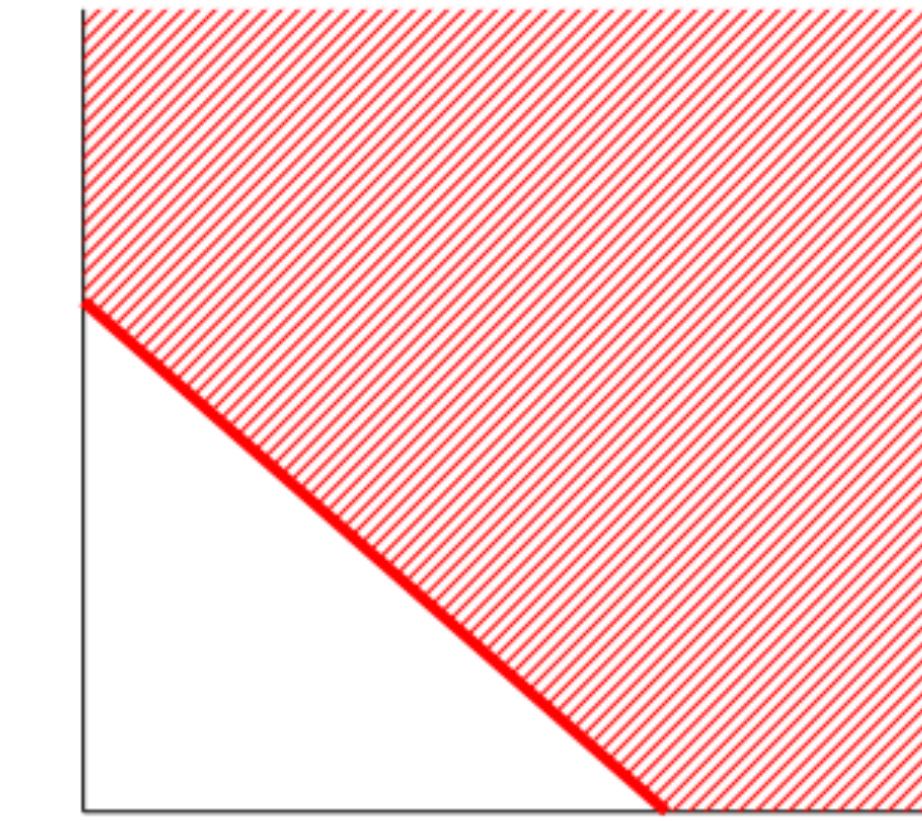
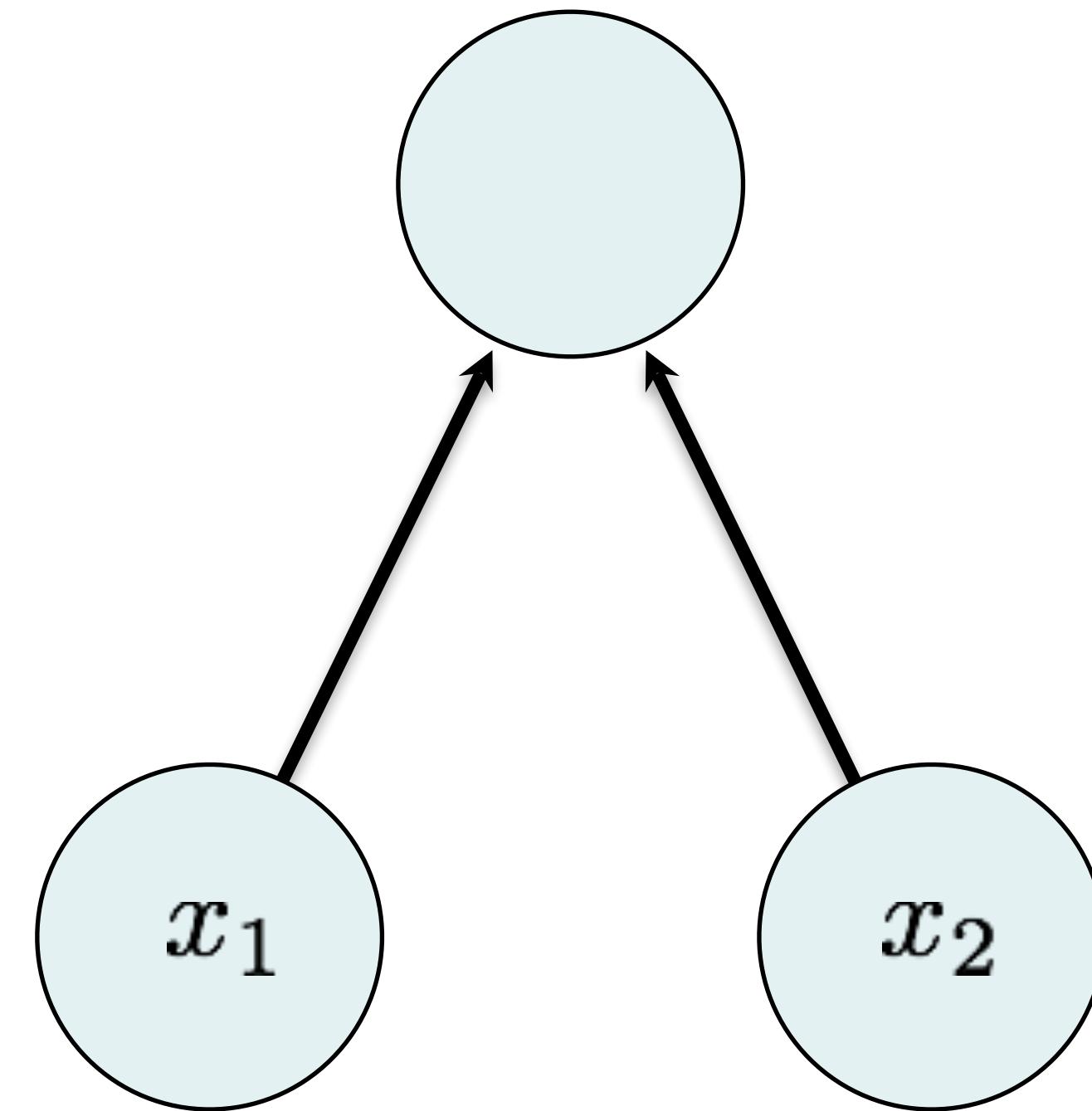
Machine learning: can work also for discrete inputs, strings, images, meshes, animations, ...

Learning a Simple Separator/Classifier



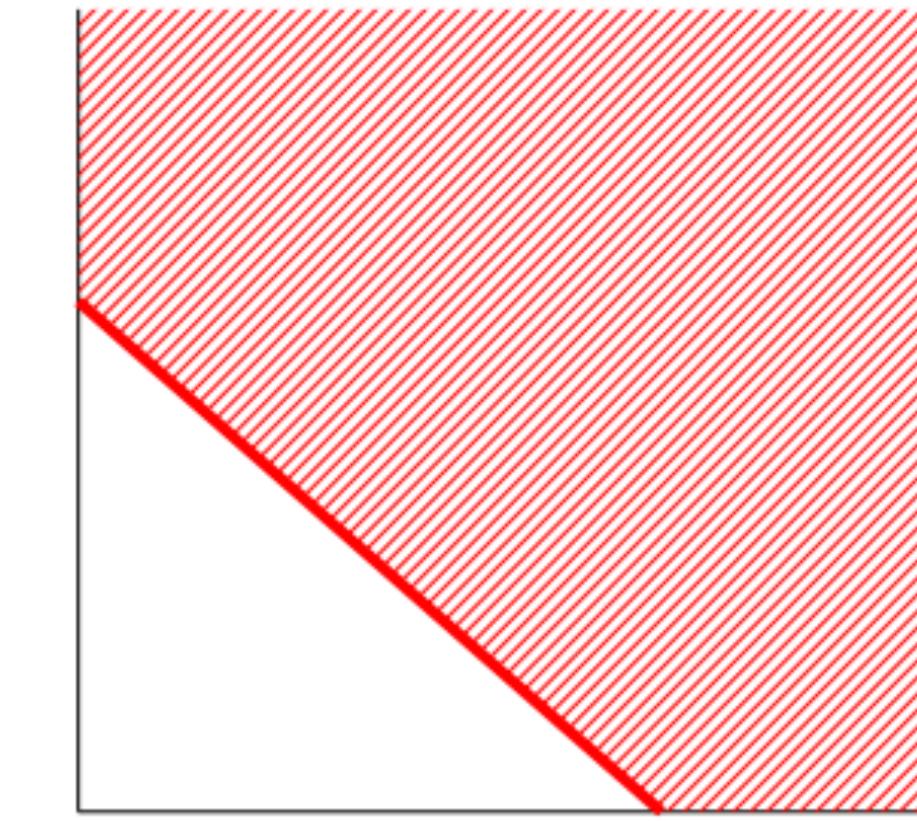
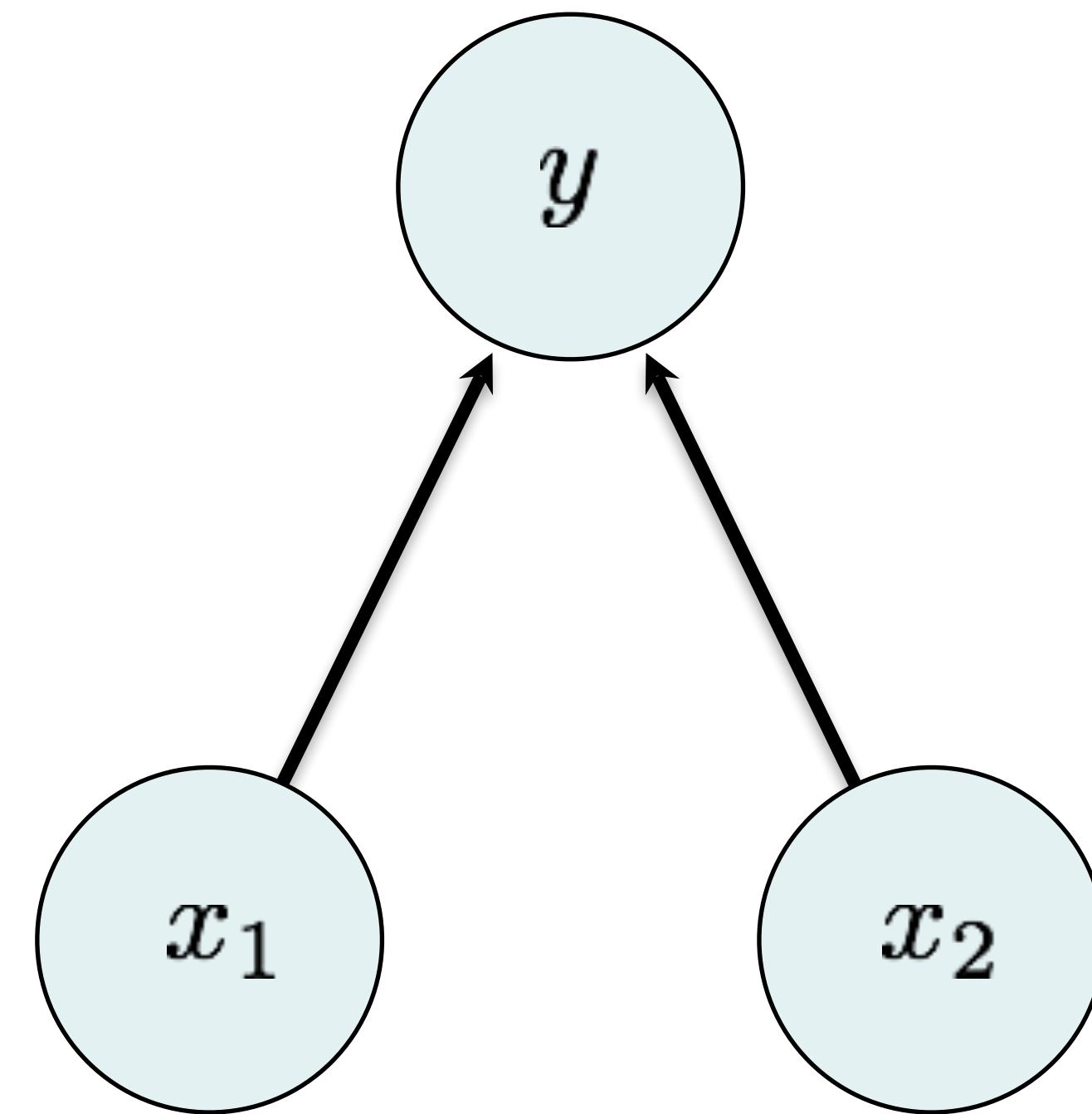
separating hyperplane

Learning a Simple Separator/Classifier



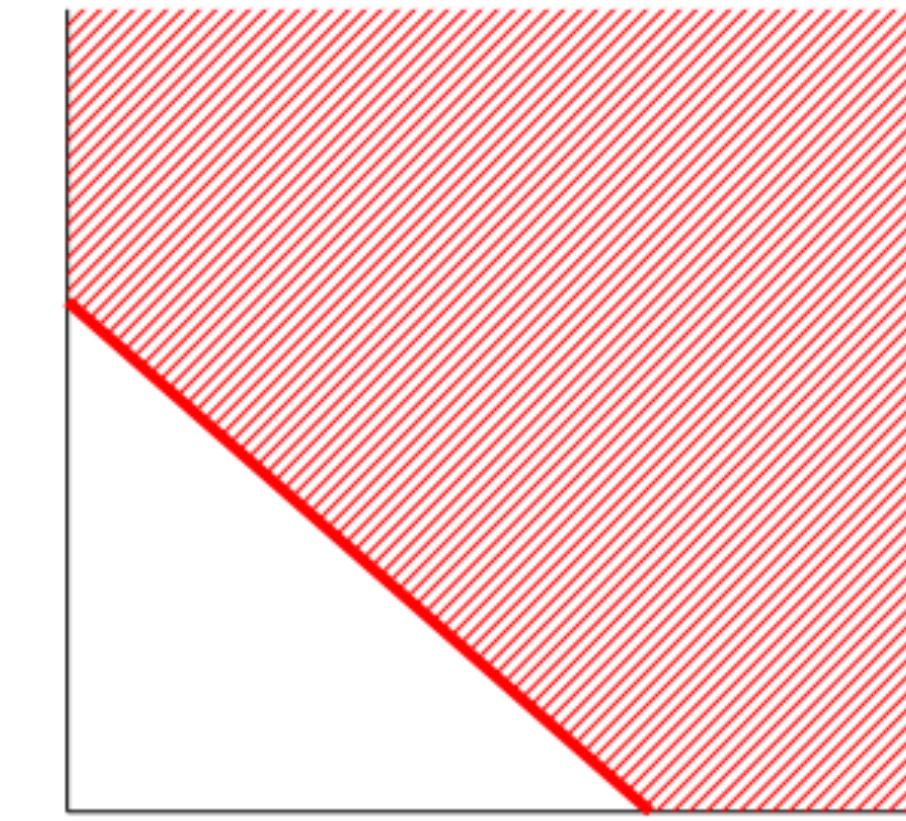
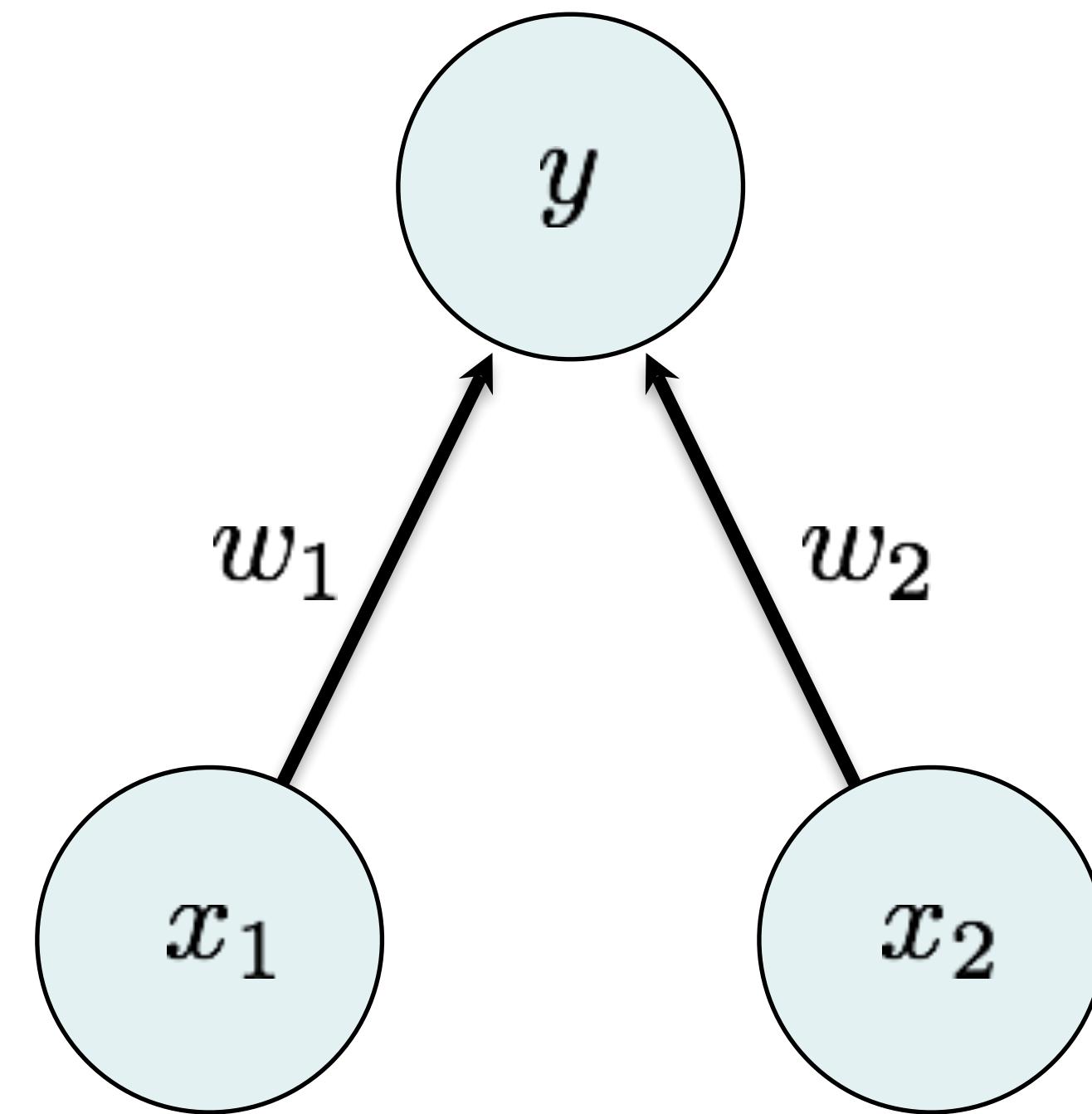
separating hyperplane

Learning a Simple Separator/Classifier



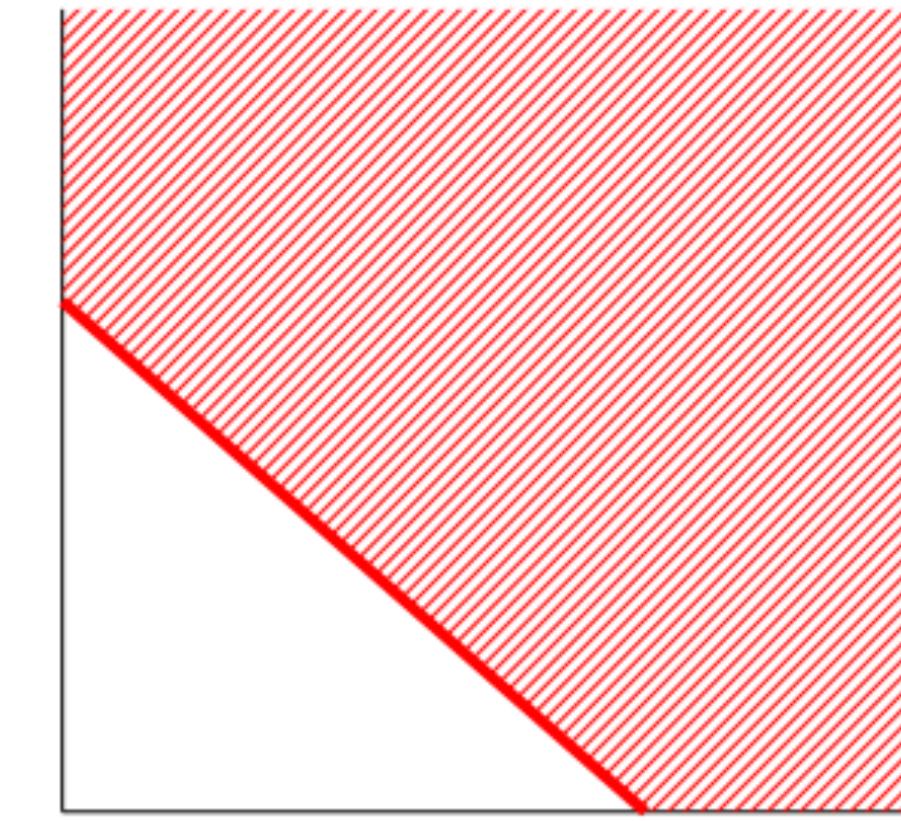
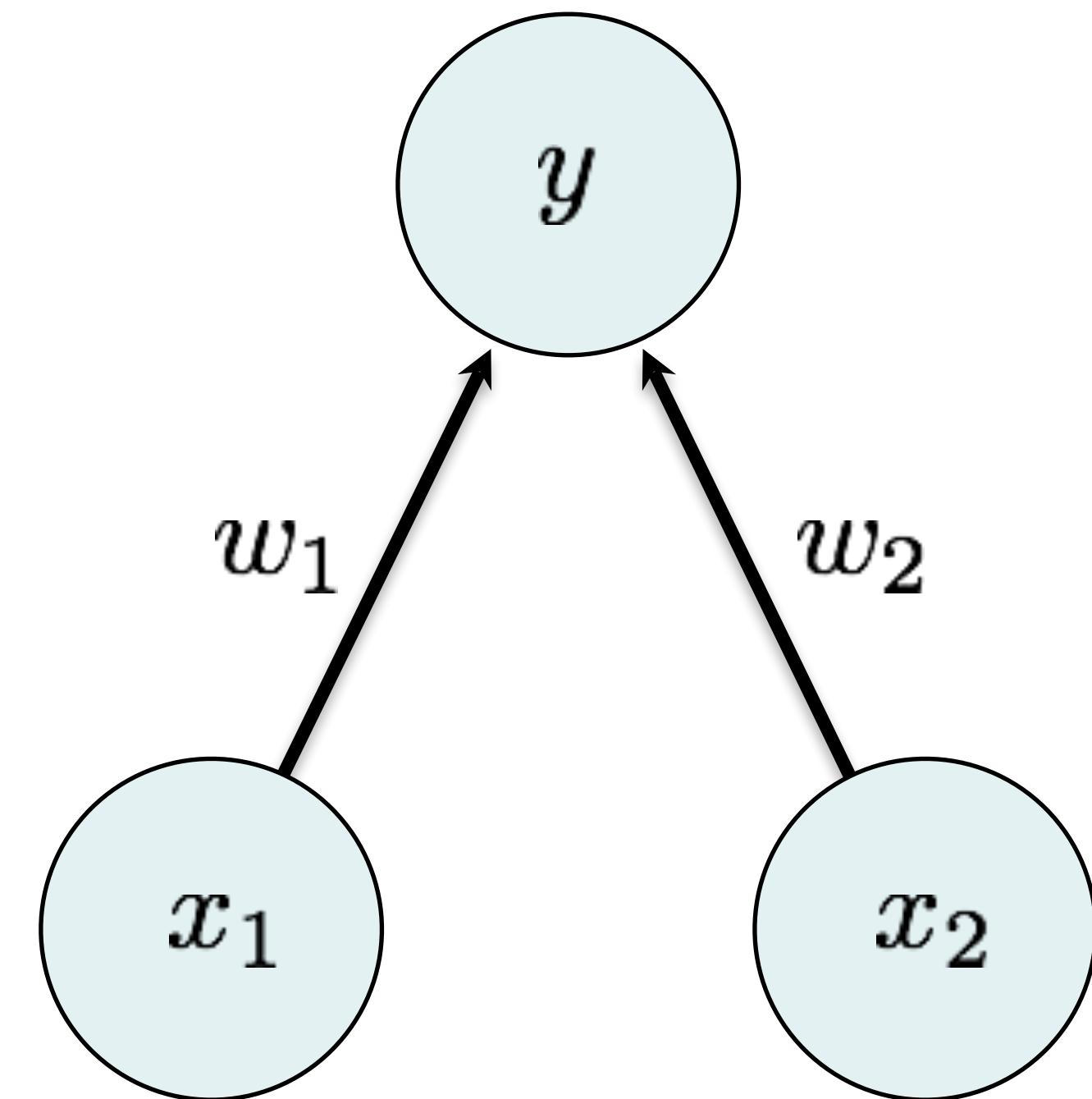
separating hyperplane

Learning a Simple Separator/Classifier



separating hyperplane

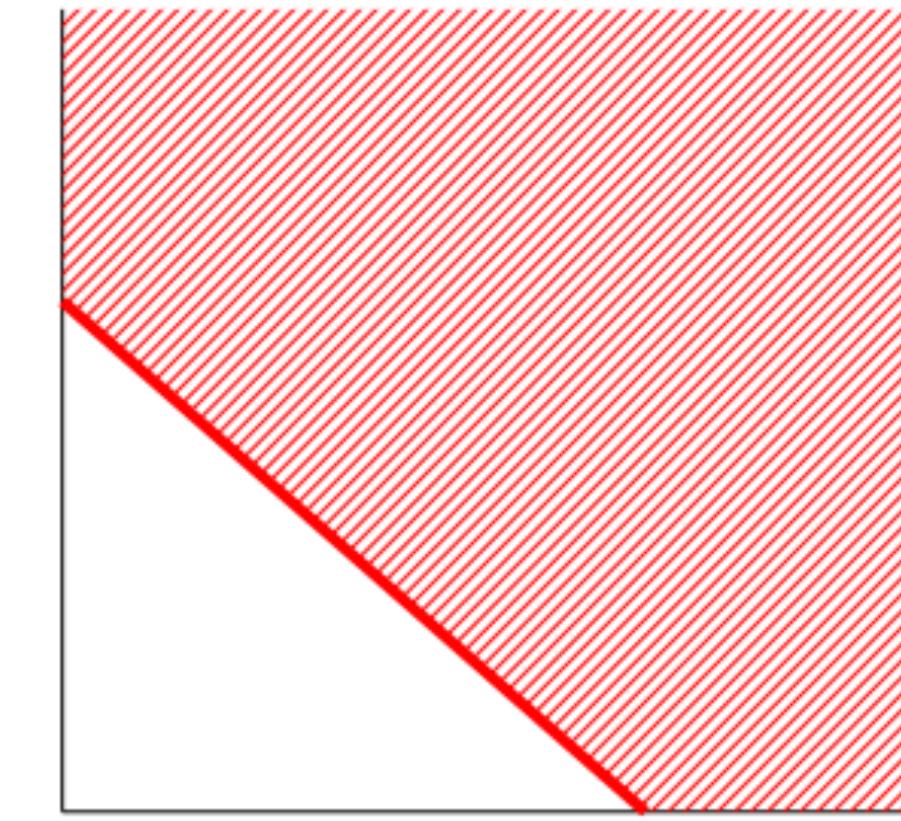
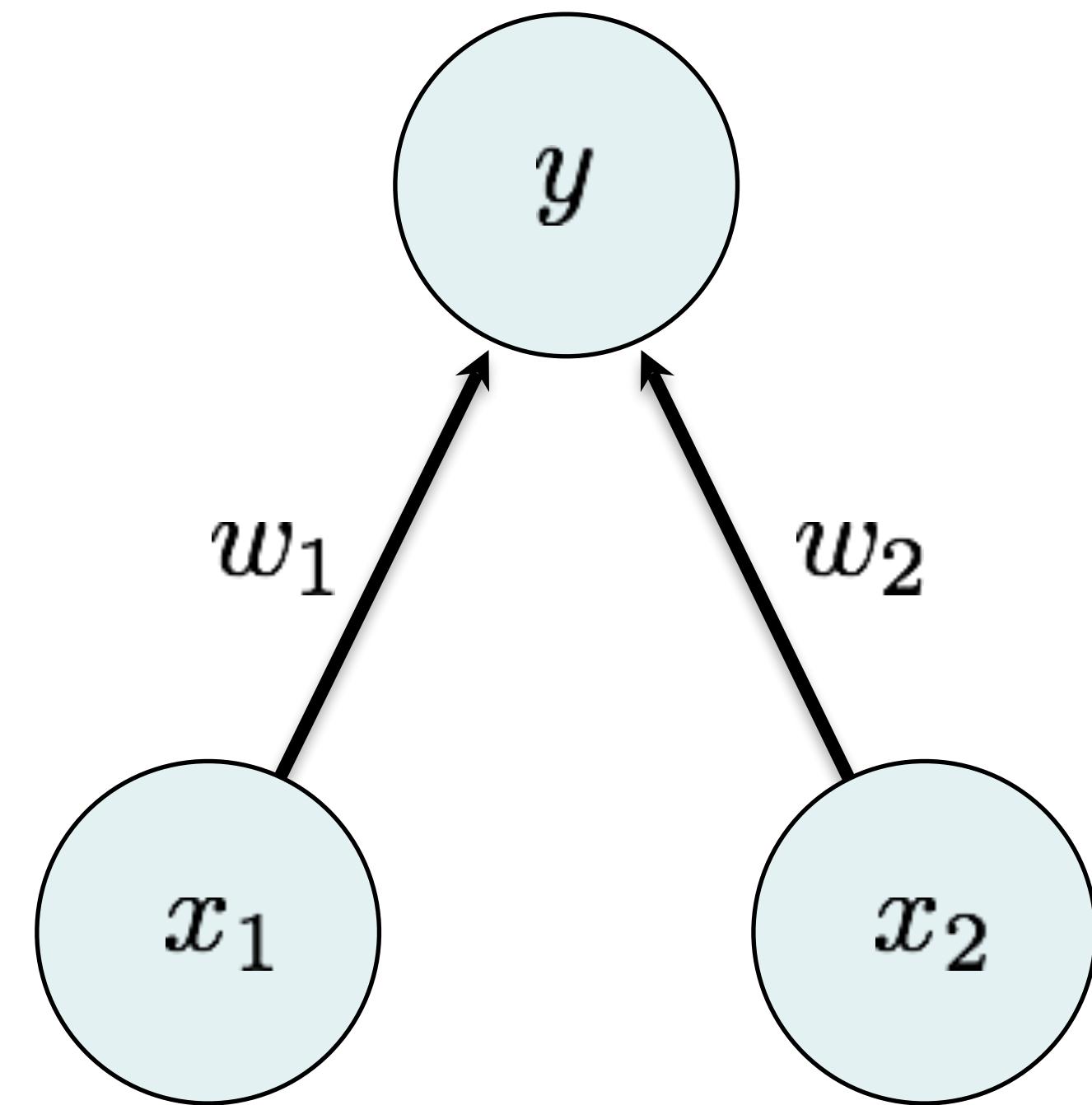
Learning a Simple Separator/Classifier



separating hyperplane

$$y = f(w_1x_1 + w_2x_2)$$

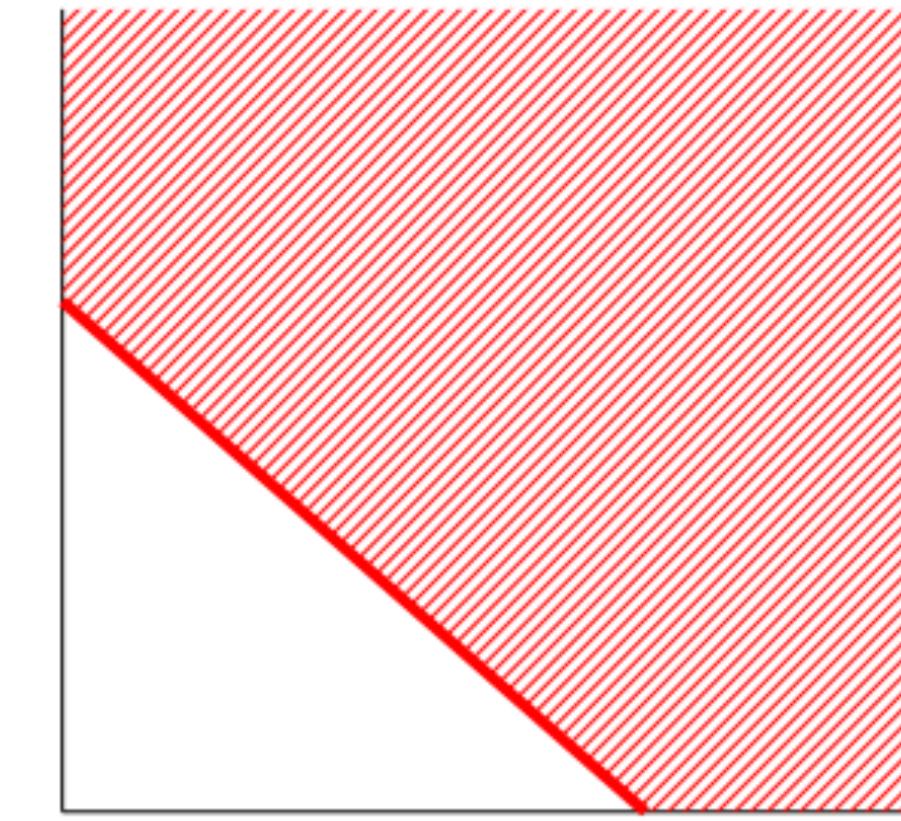
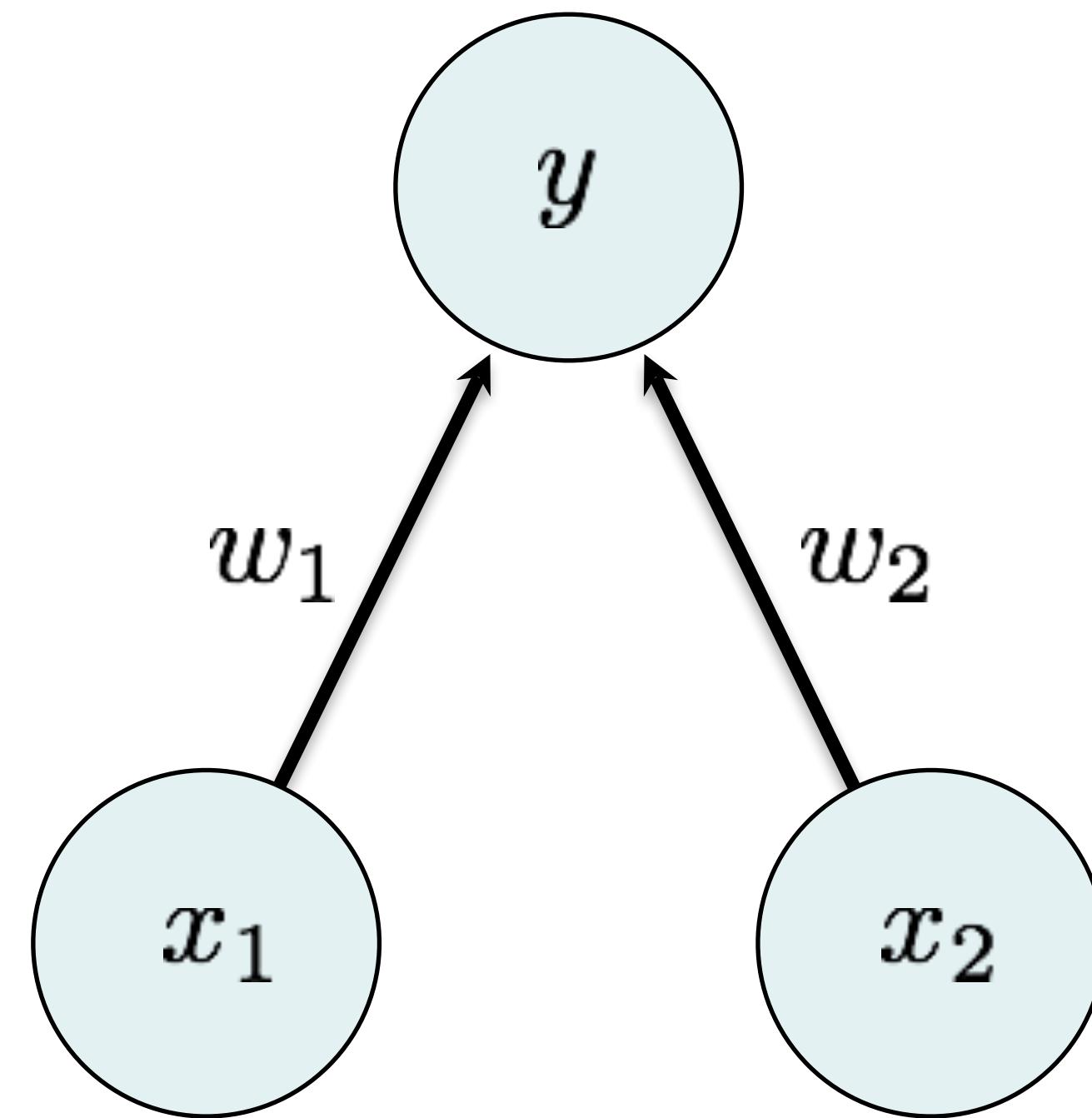
Learning a Simple Separator/Classifier



separating hyperplane

$$y = f(w_1x_1 + w_2x_2) = \mathcal{H}(w_1x_1 + w_2x_2)$$

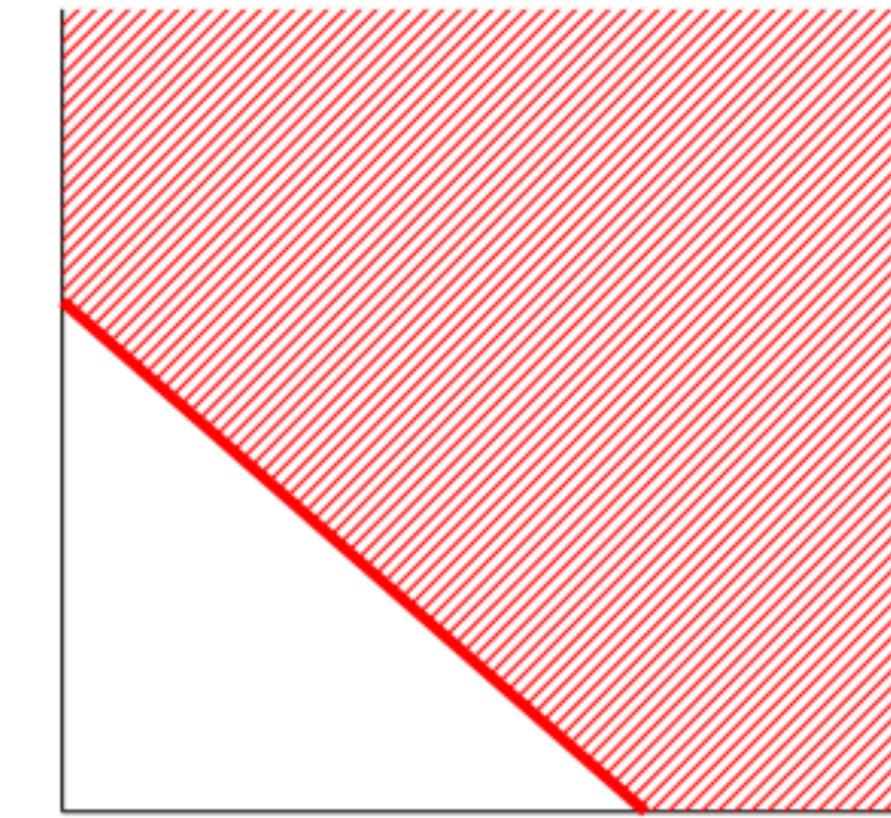
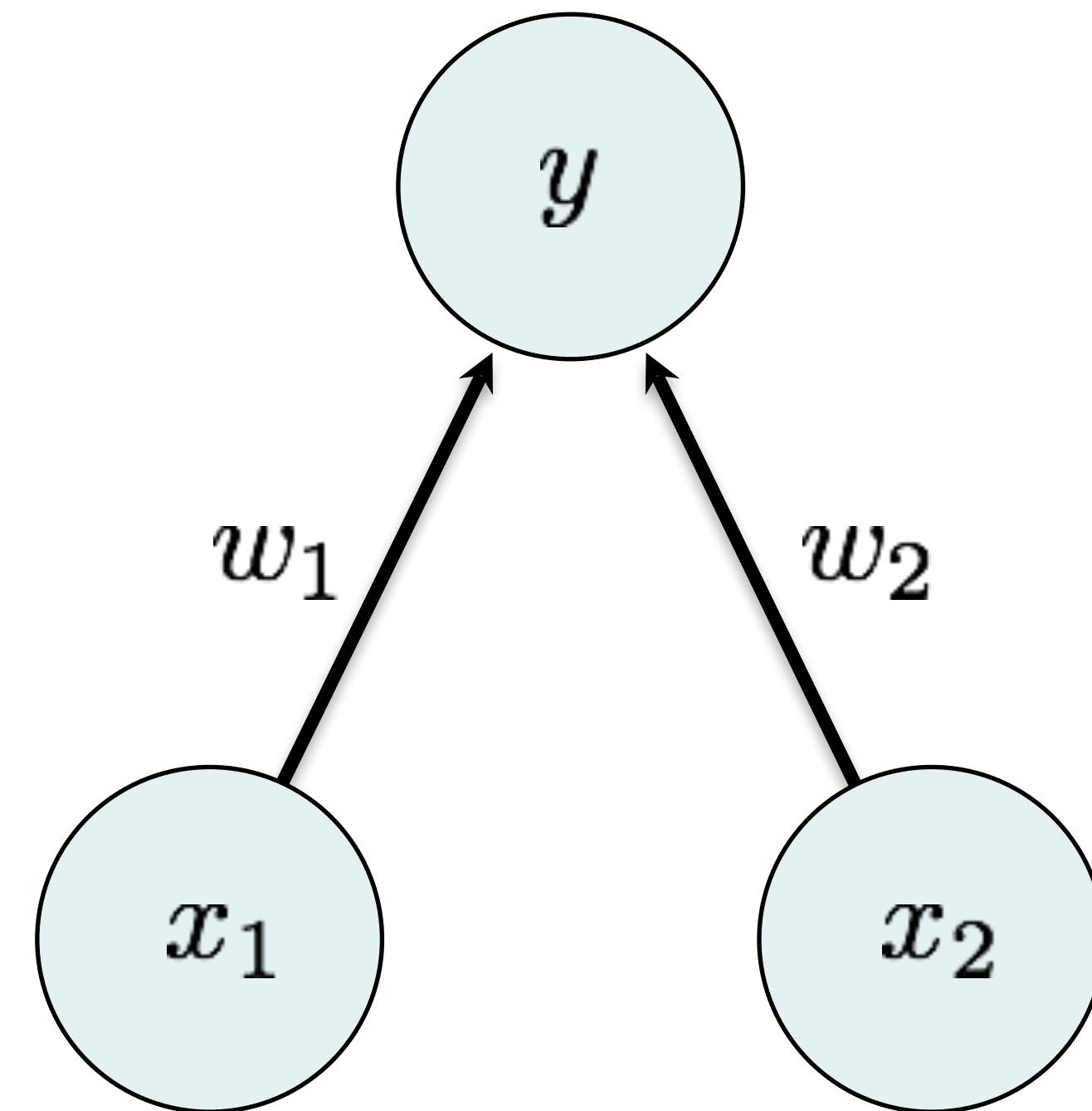
Learning a Simple Separator/Classifier



**separating hyperplane
fixed non-linearity**

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

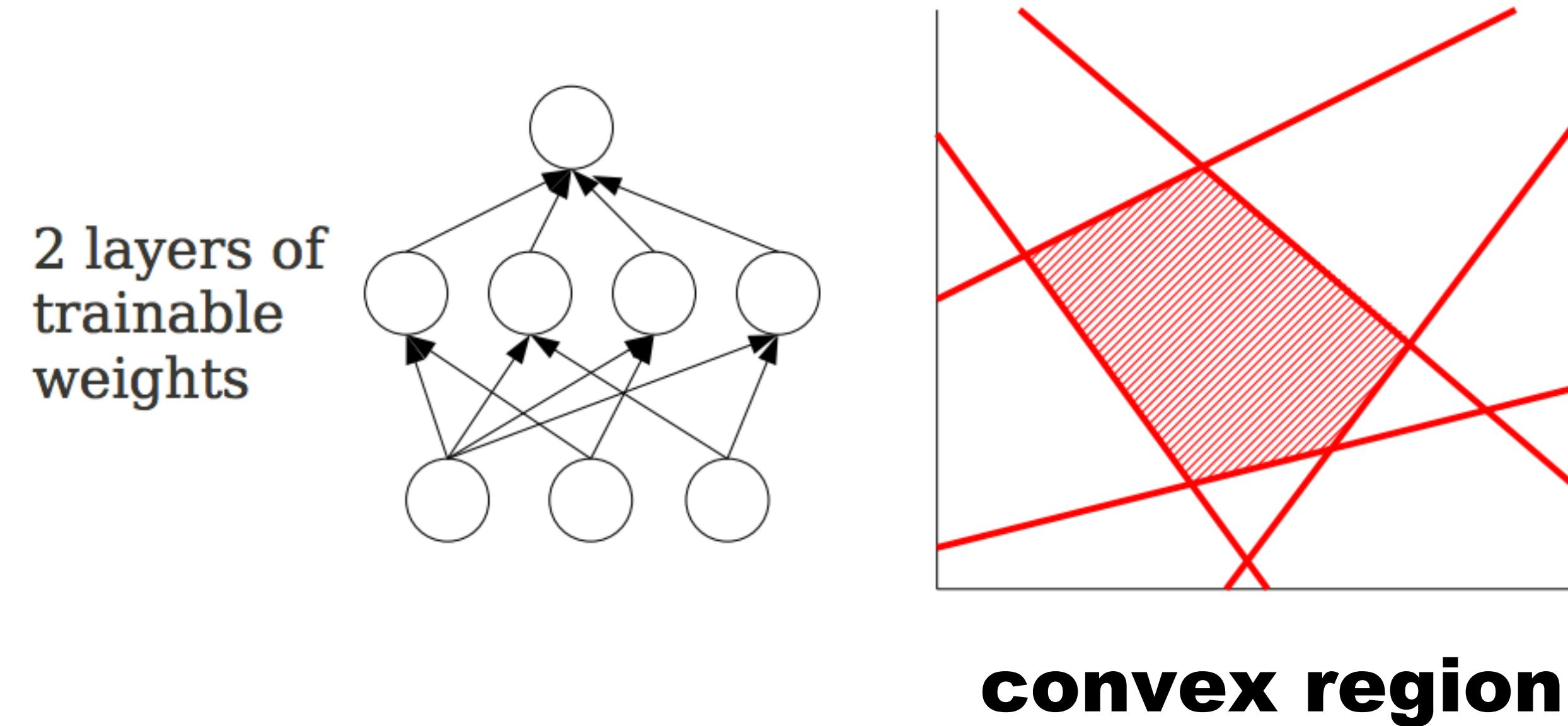
Learning a Simple Separator/Classifier



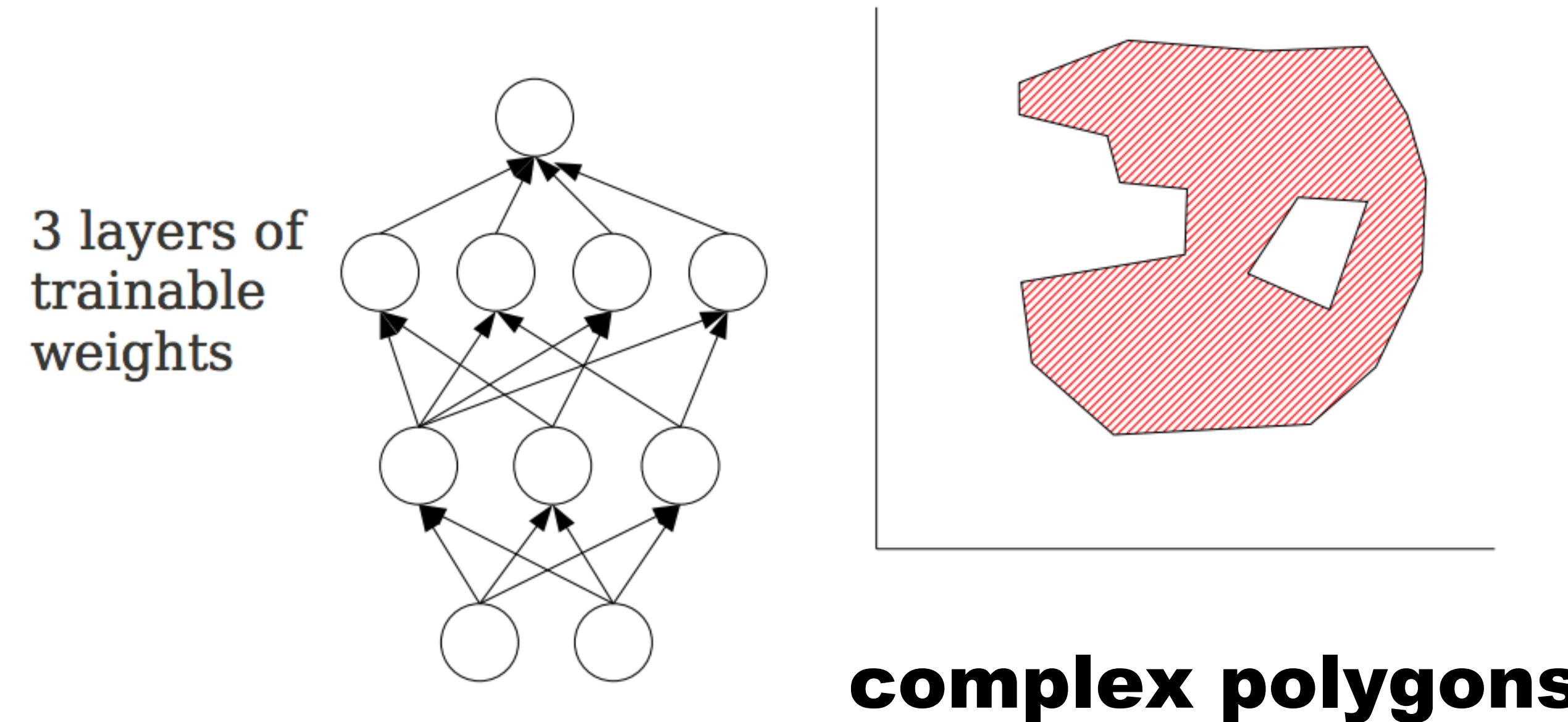
**separating hyperplane
fixed non-linearity**

$$y = f(w_1x_1 + w_2x_2) = \mathcal{H}(w_1x_1 + w_2x_2)$$

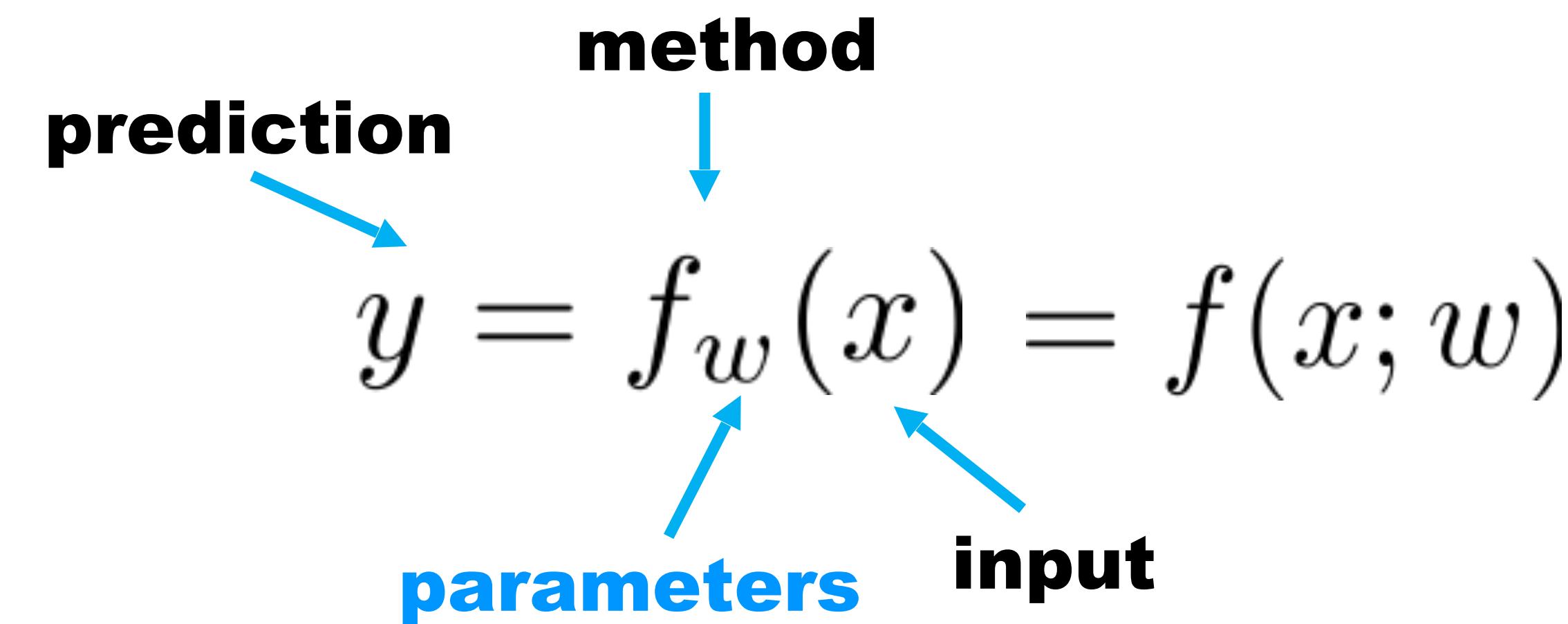
Combining Simple Functions/Classifiers



Combining Simple Functions/Classifiers



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

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

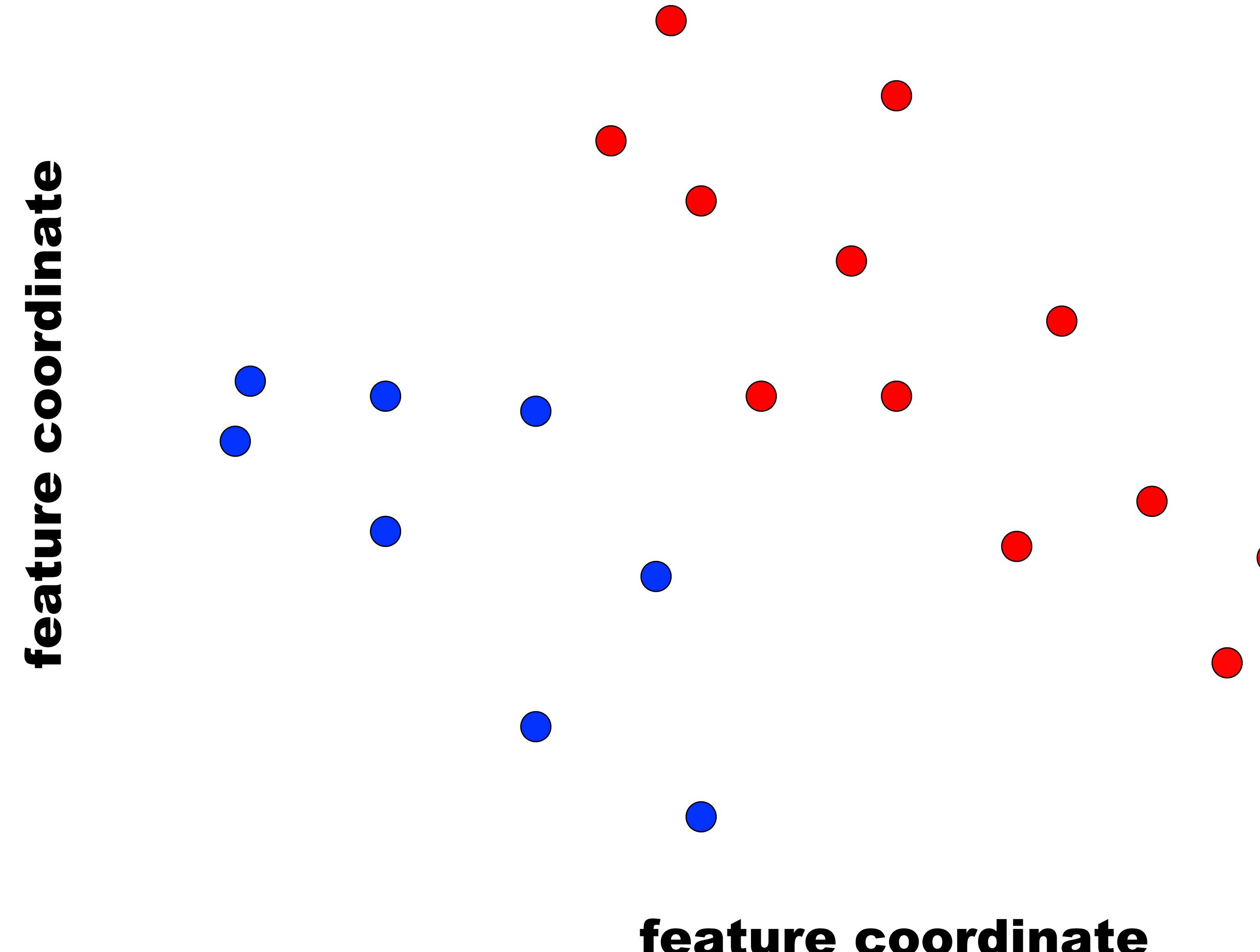
Assumption: Linear Function

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

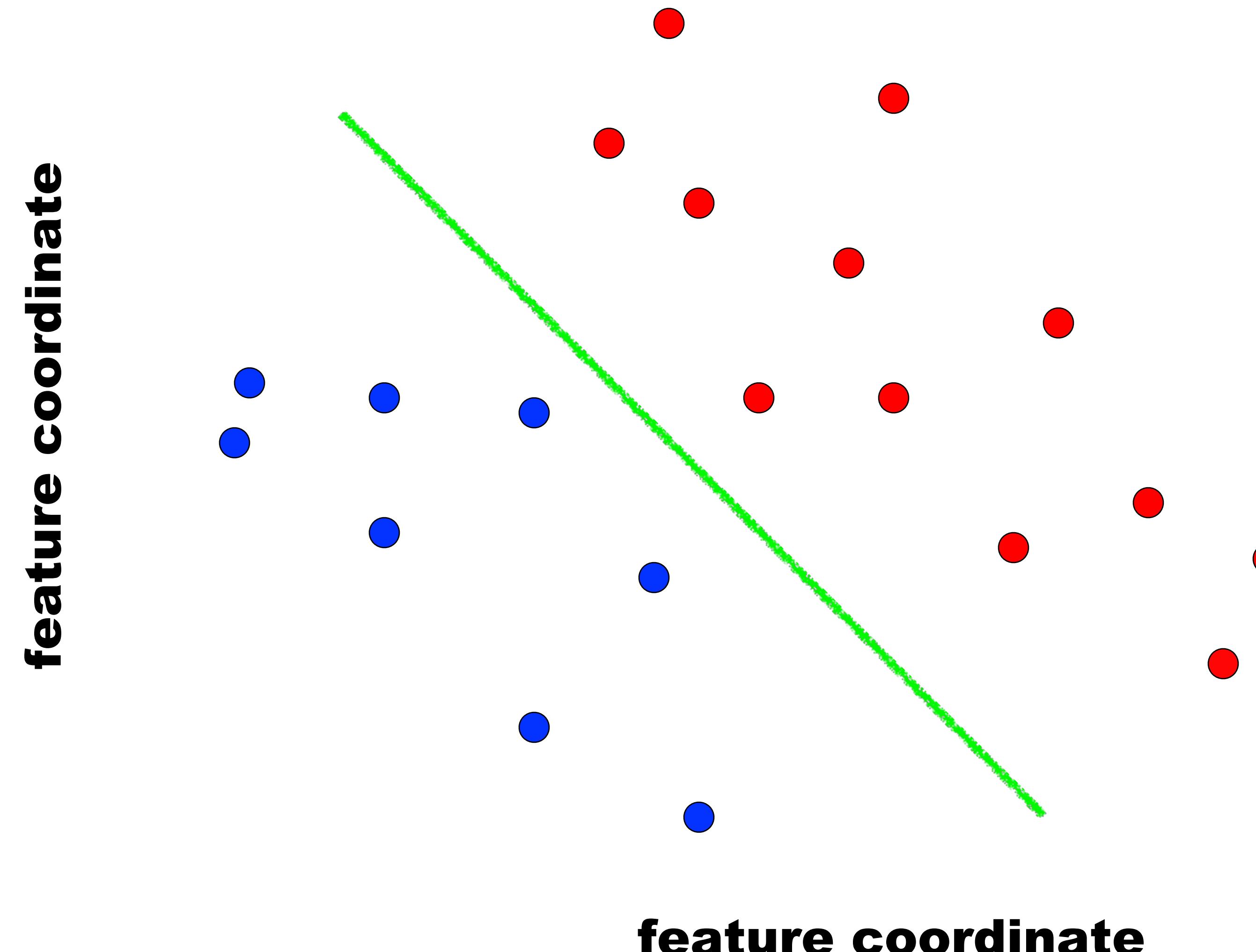
$$\mathbf{w}^T \mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{d=1}^D \mathbf{w}_d \mathbf{x}_d$$

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

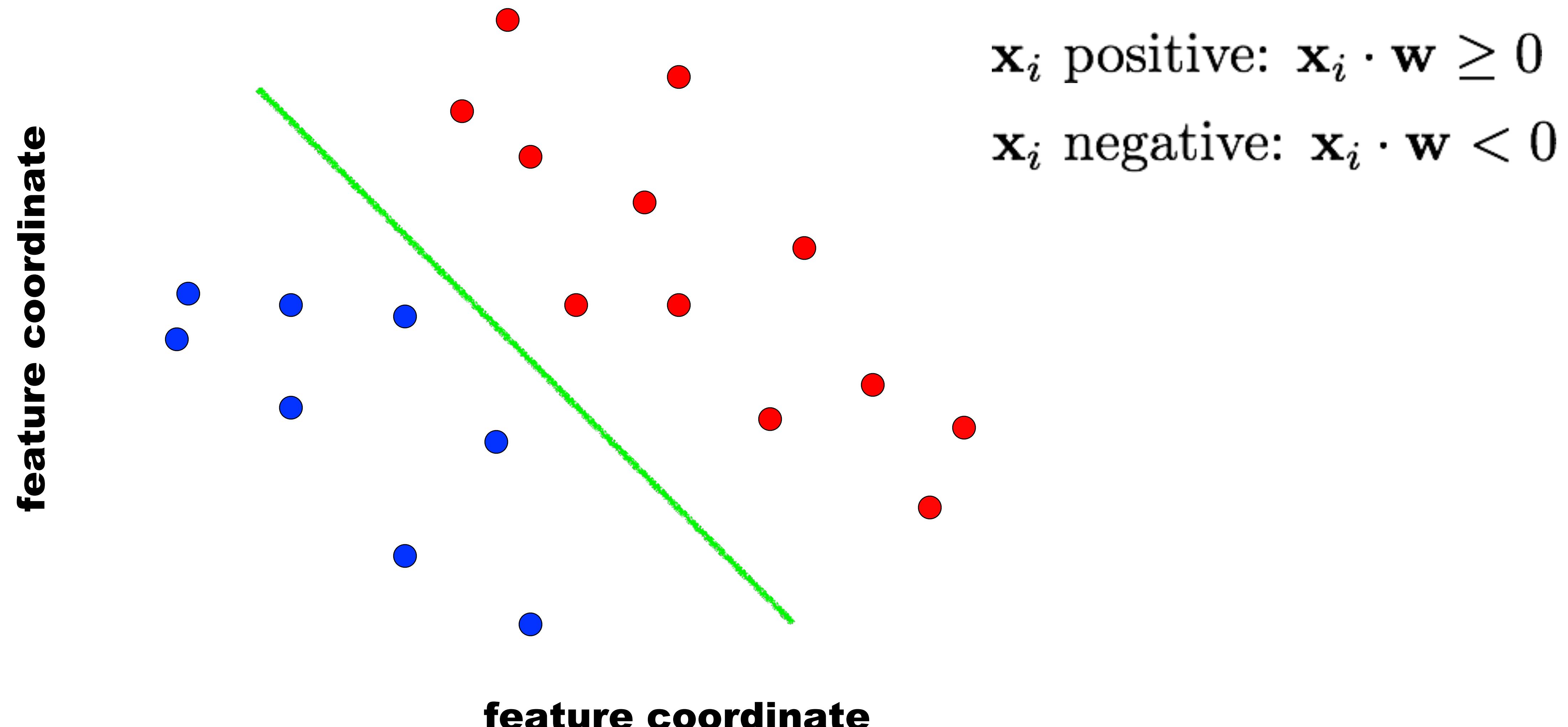
Reminder: Linear Classifier



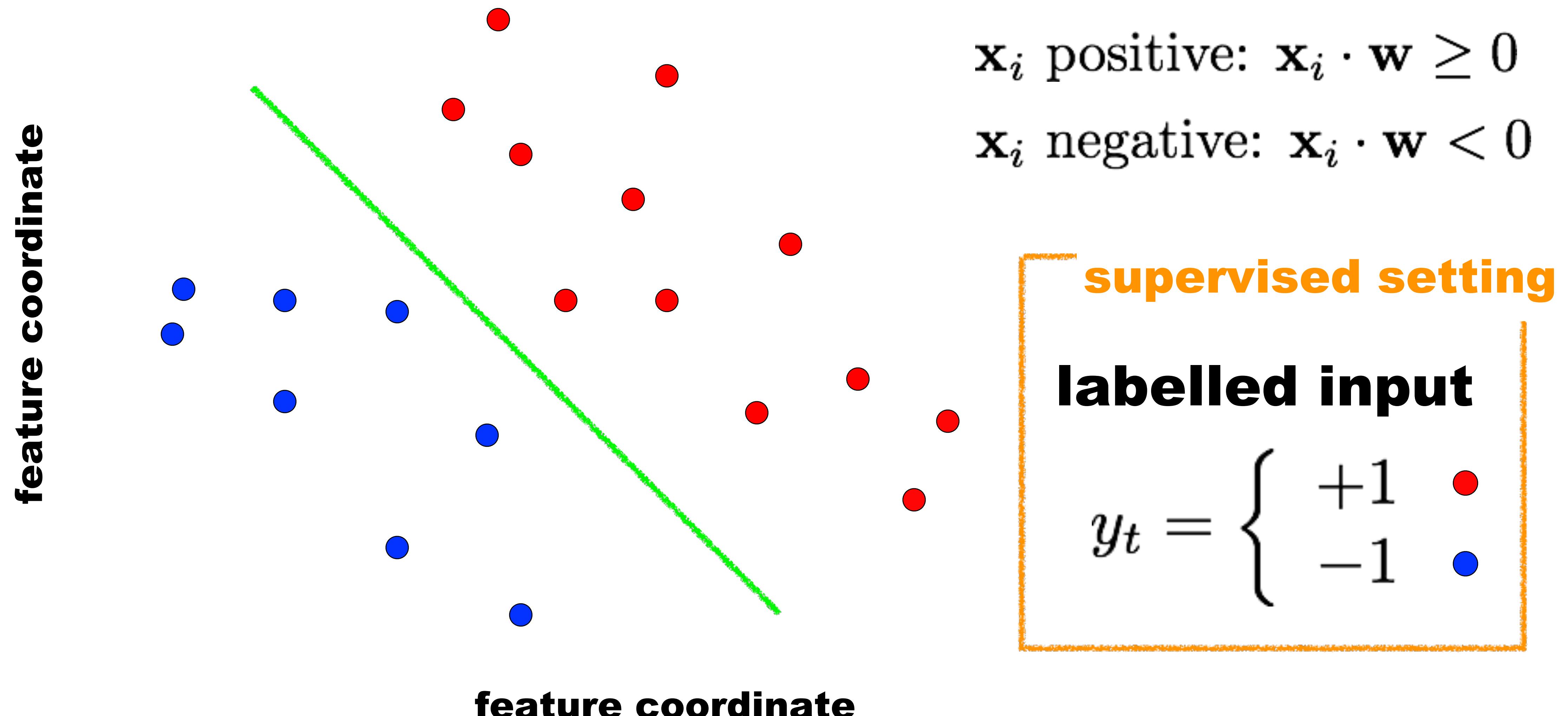
Reminder: Linear Classifier



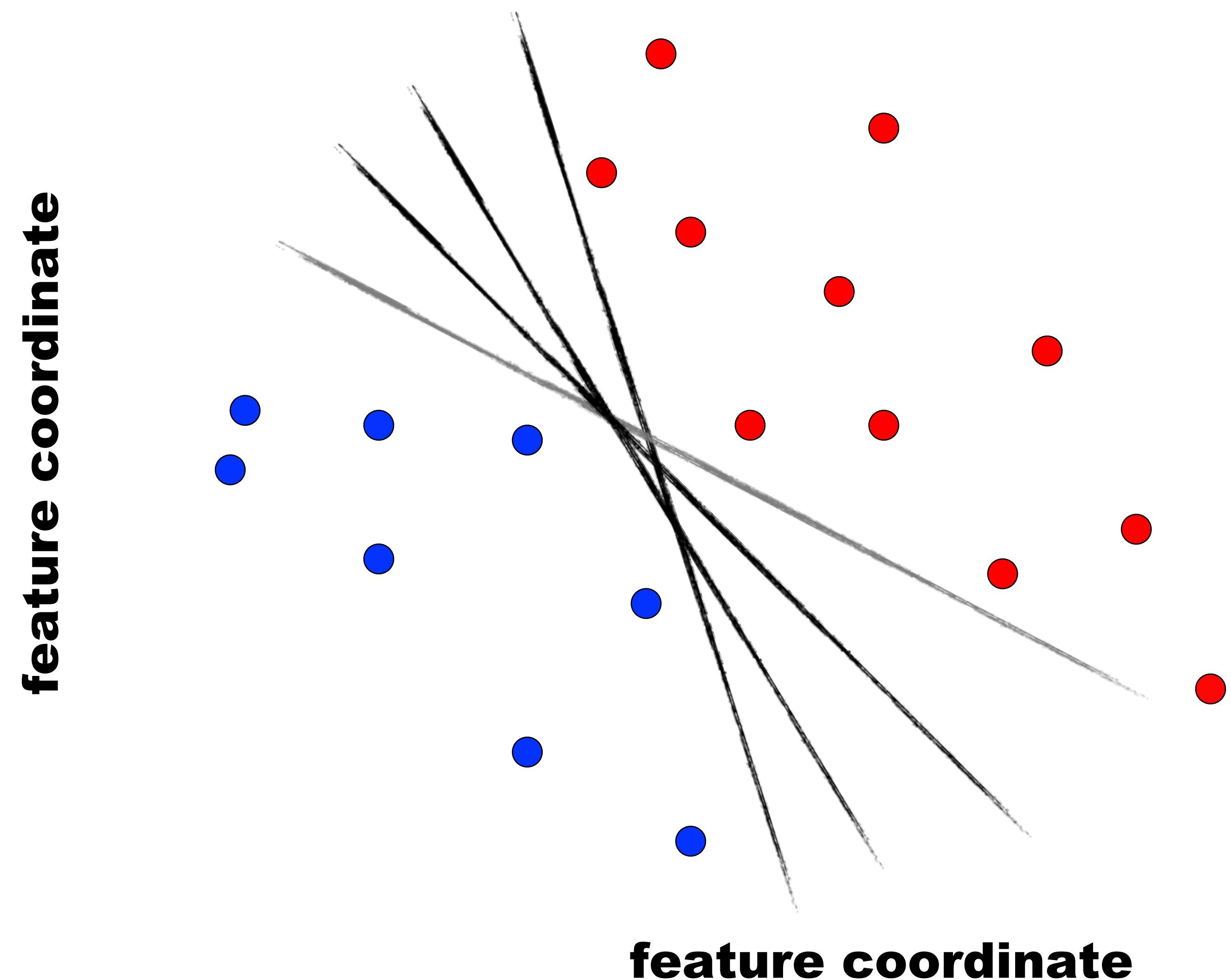
Reminder: Linear Classifier



Reminder: Linear Classifier



Which Line to Pick?



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

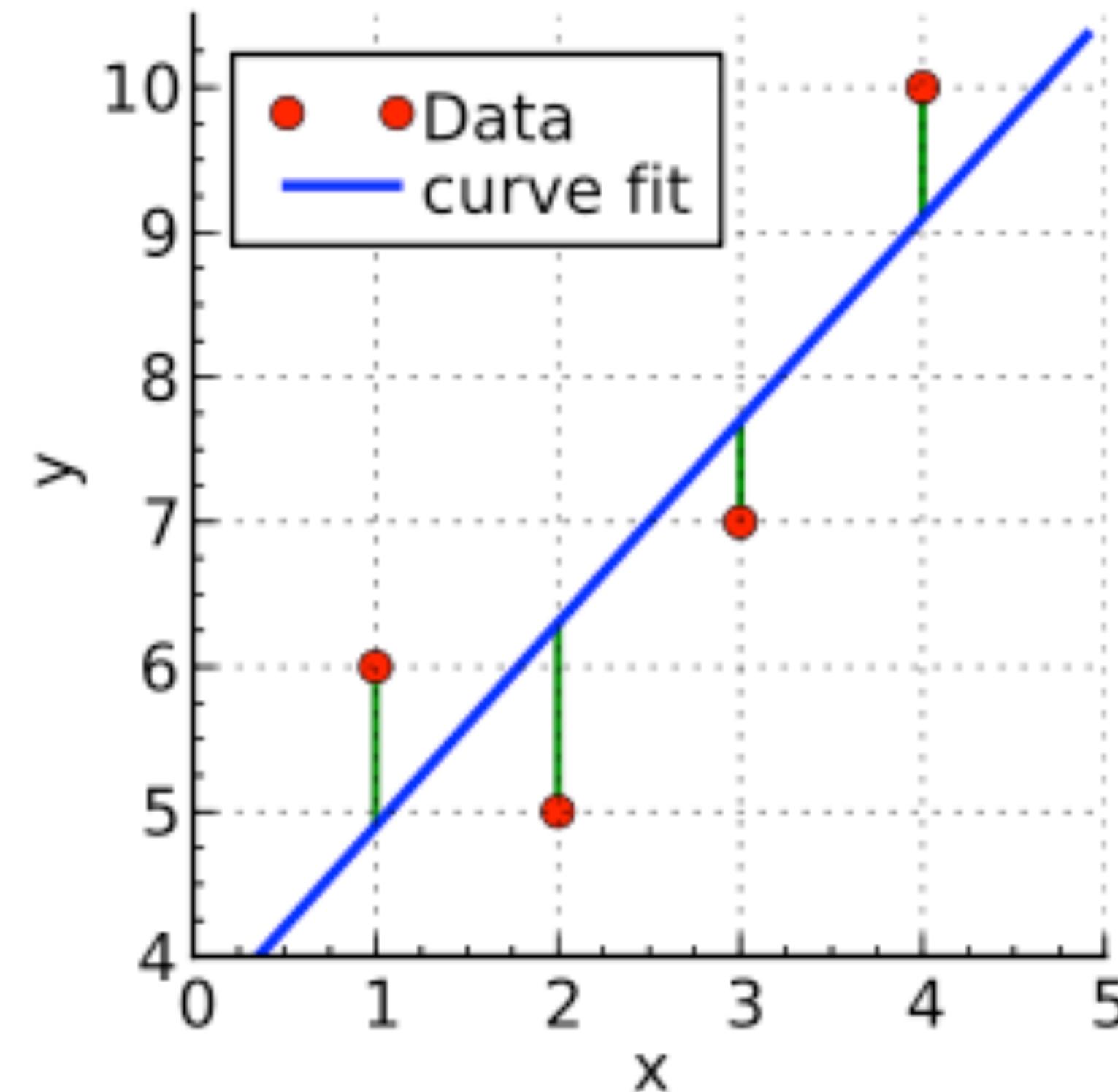
\mathbf{x}_i negative: $\mathbf{x}_i \cdot \mathbf{w} < 0$

supervised setting

labelled input

$$y_t = \begin{cases} +1 & \text{red dot} \\ -1 & \text{blue dot} \end{cases}$$

Linear Regression in 1D

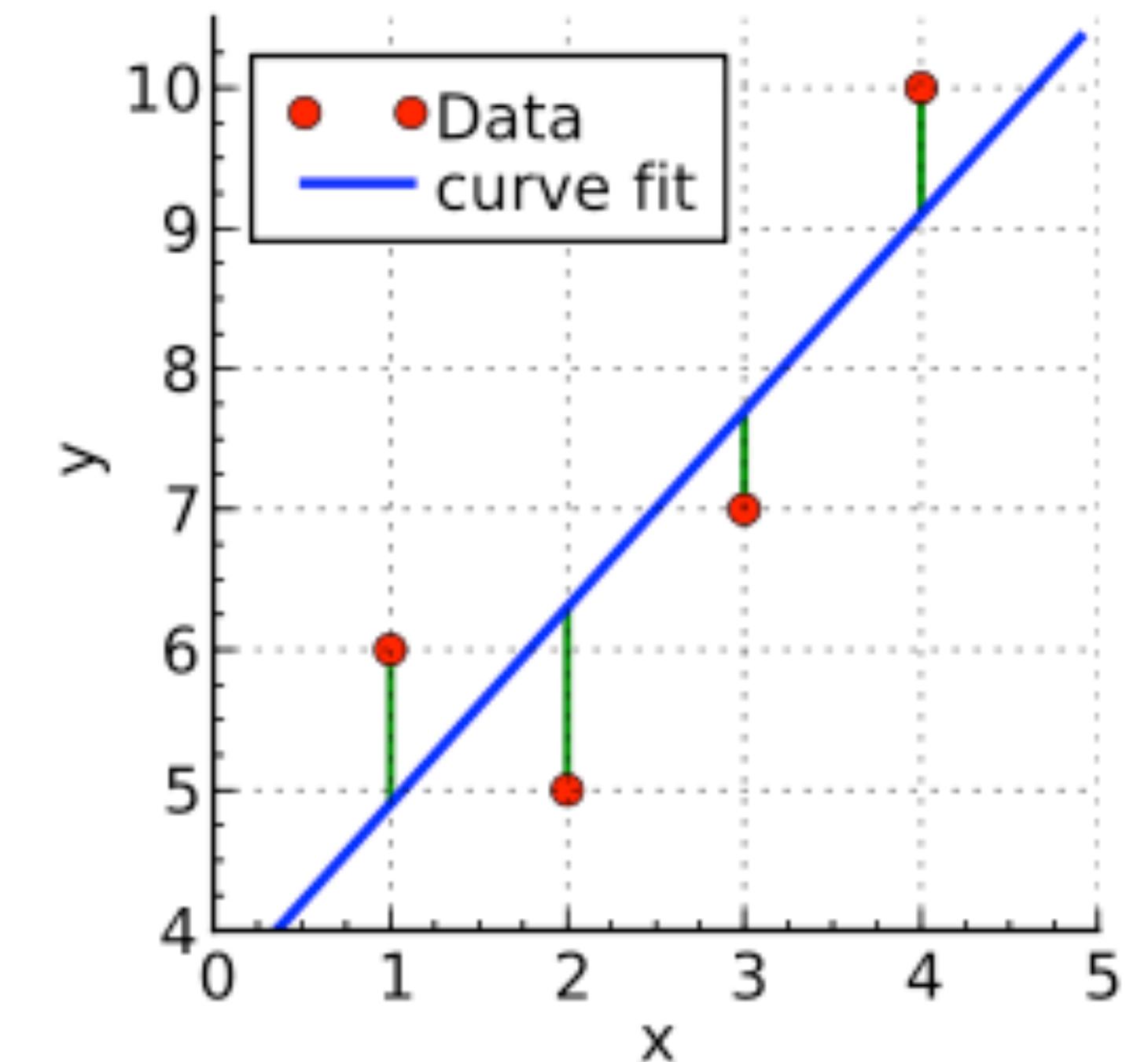


Training set:
input–output pairs

$$\mathcal{S} = \{(x^i, y^i)\}, \quad i = 1 \dots, N$$
$$x^i \in \mathbb{R}, \quad y^i \in \mathbb{R}$$

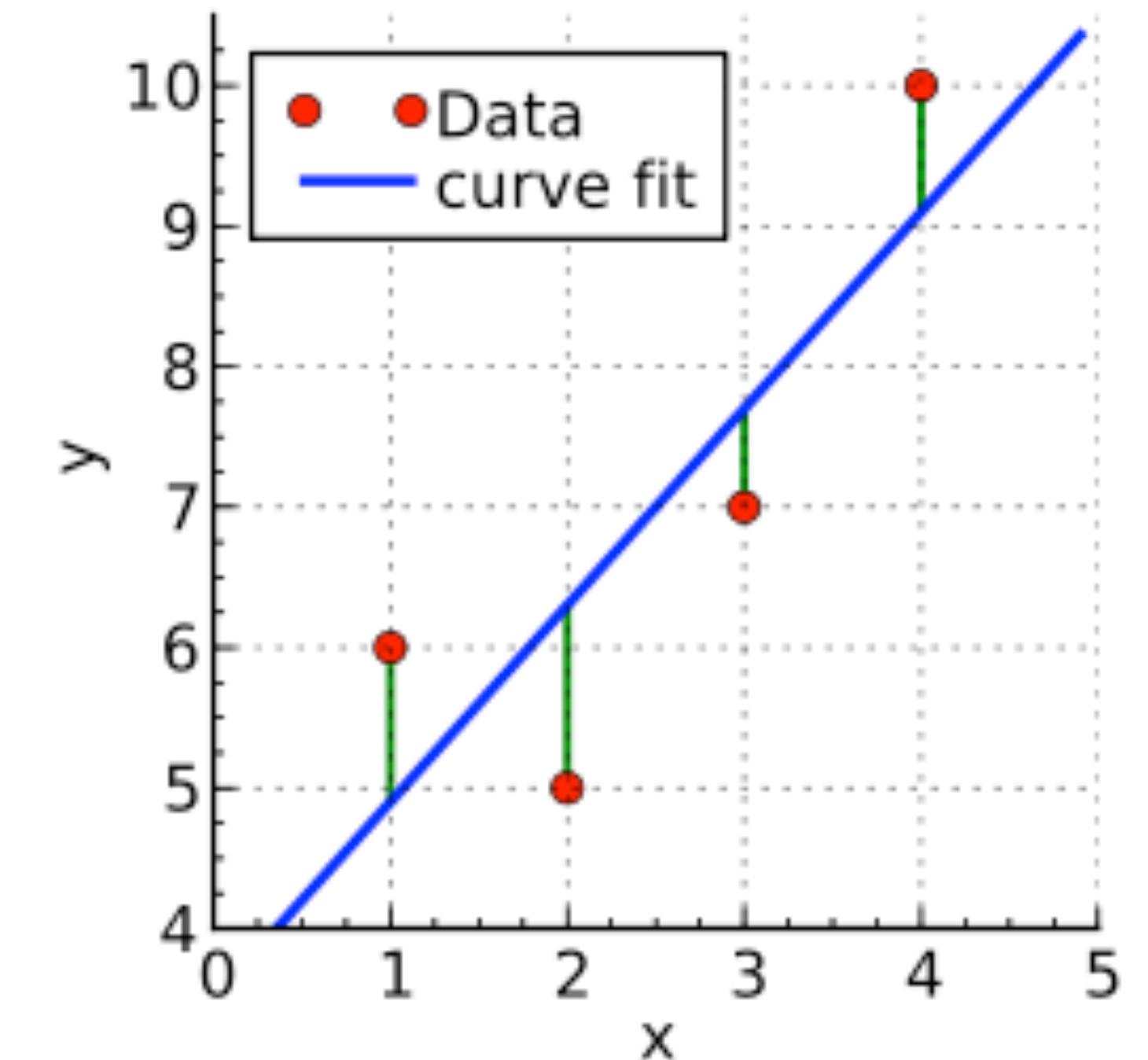
Linear regression in 1D

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



Linear regression in 1D

$$\begin{aligned}y^i &= w_0 + w_1 x_1^i + \epsilon^i \\&= w_0 x_0^i + w_1 x_1^i + \epsilon^i, \quad x_0^i = 1, \quad \forall i\end{aligned}$$

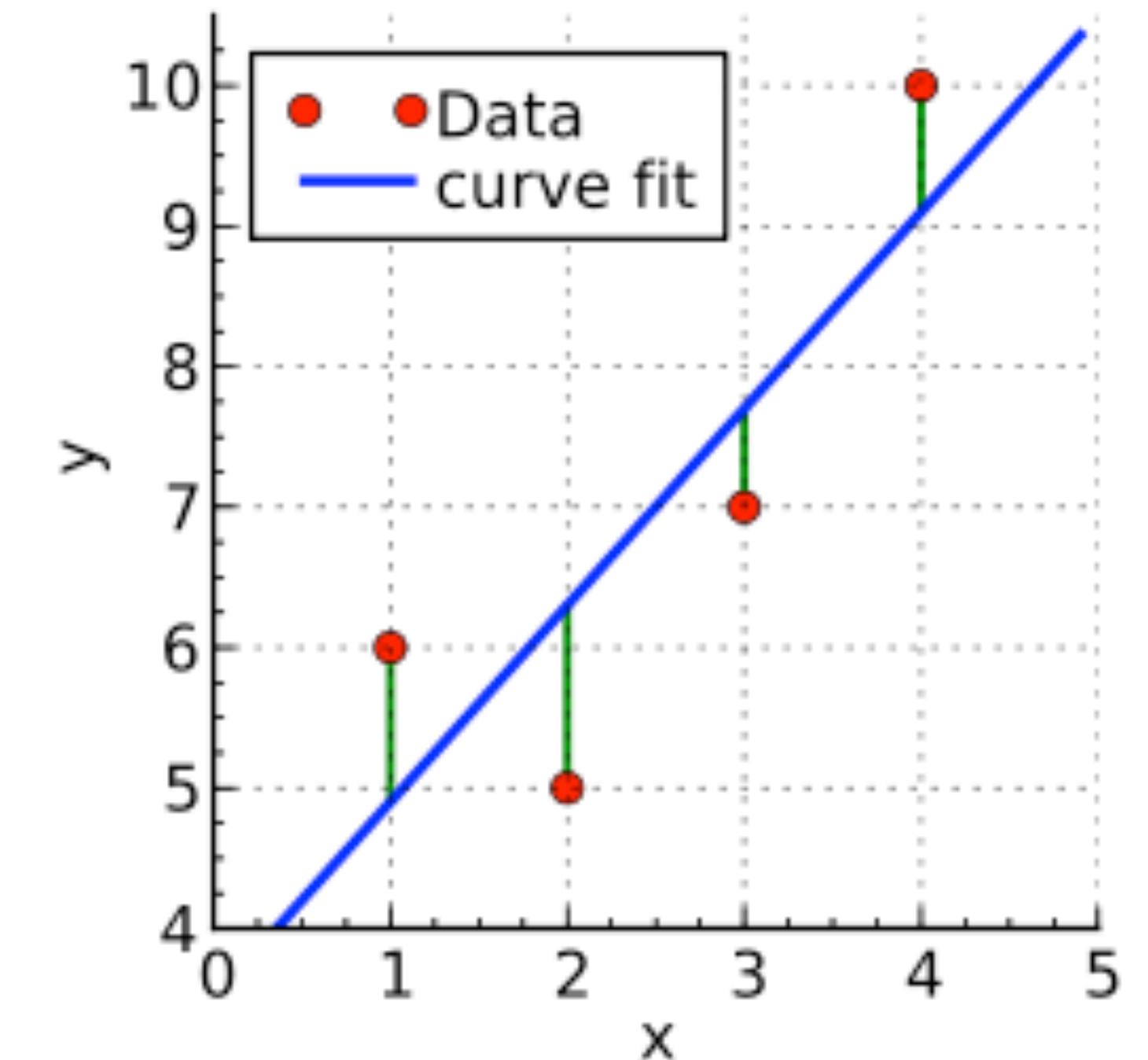


Linear regression in 1D

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

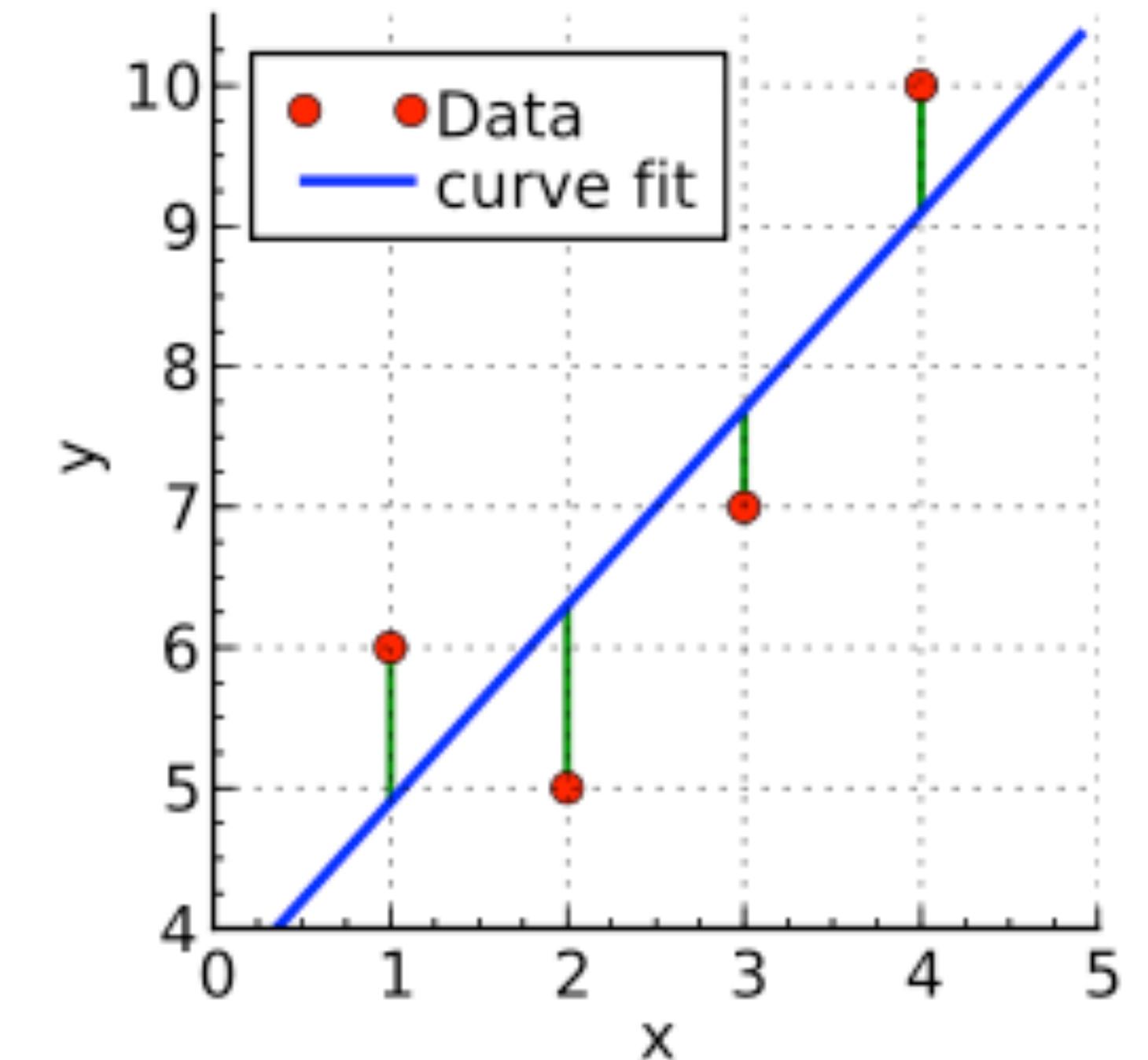
w_0 bias

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



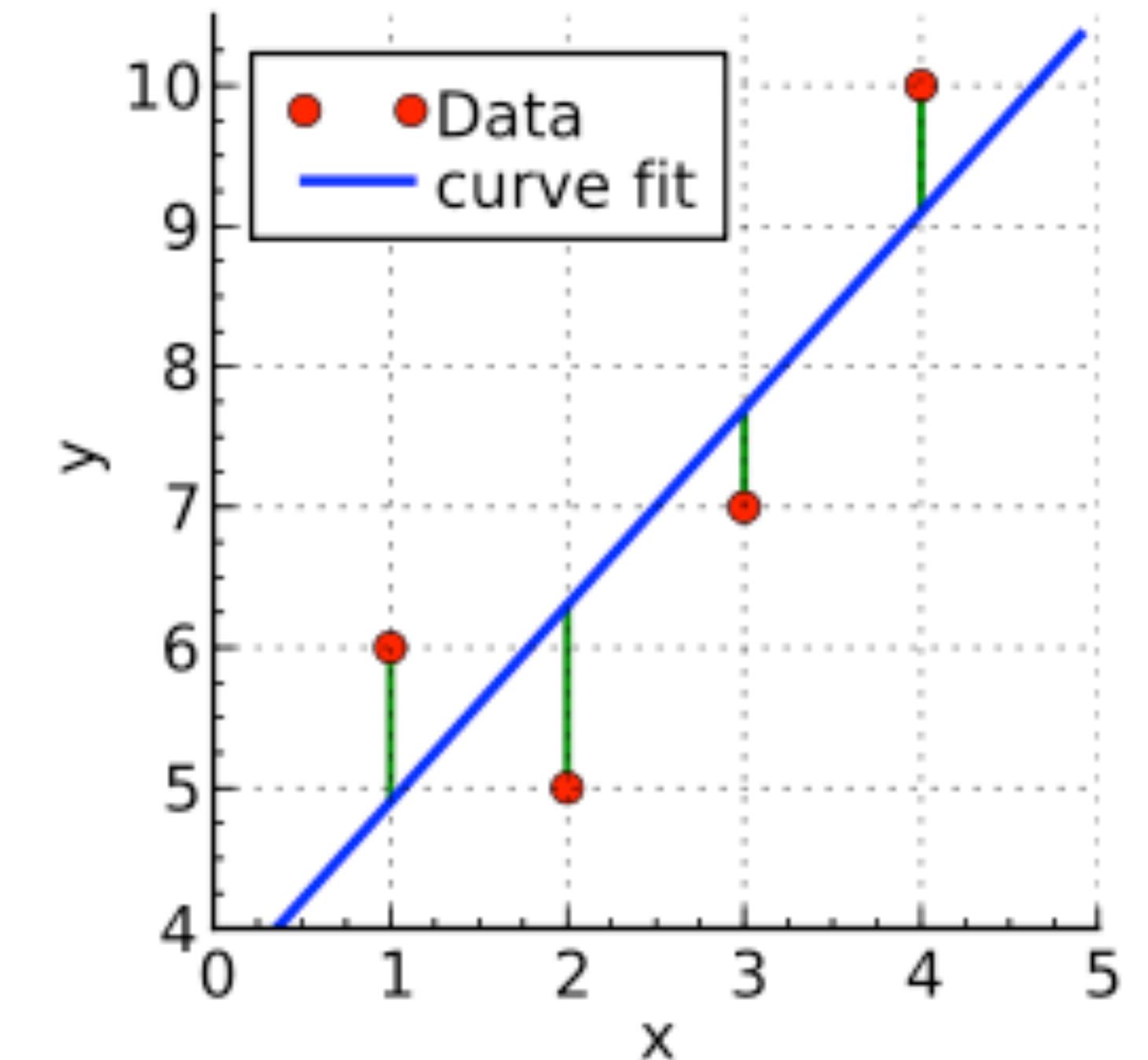
Linear regression in 1D

$$\begin{aligned}y^i &= w_0 + w_1 x_1^i + \epsilon^i && w_0 \text{ bias} \\&= w_0 x_0^i + w_1 x_1^i + \epsilon^i, & x_0^i = 1, & \forall i \\&= \mathbf{w}^T \mathbf{x}^i + \epsilon^i\end{aligned}$$



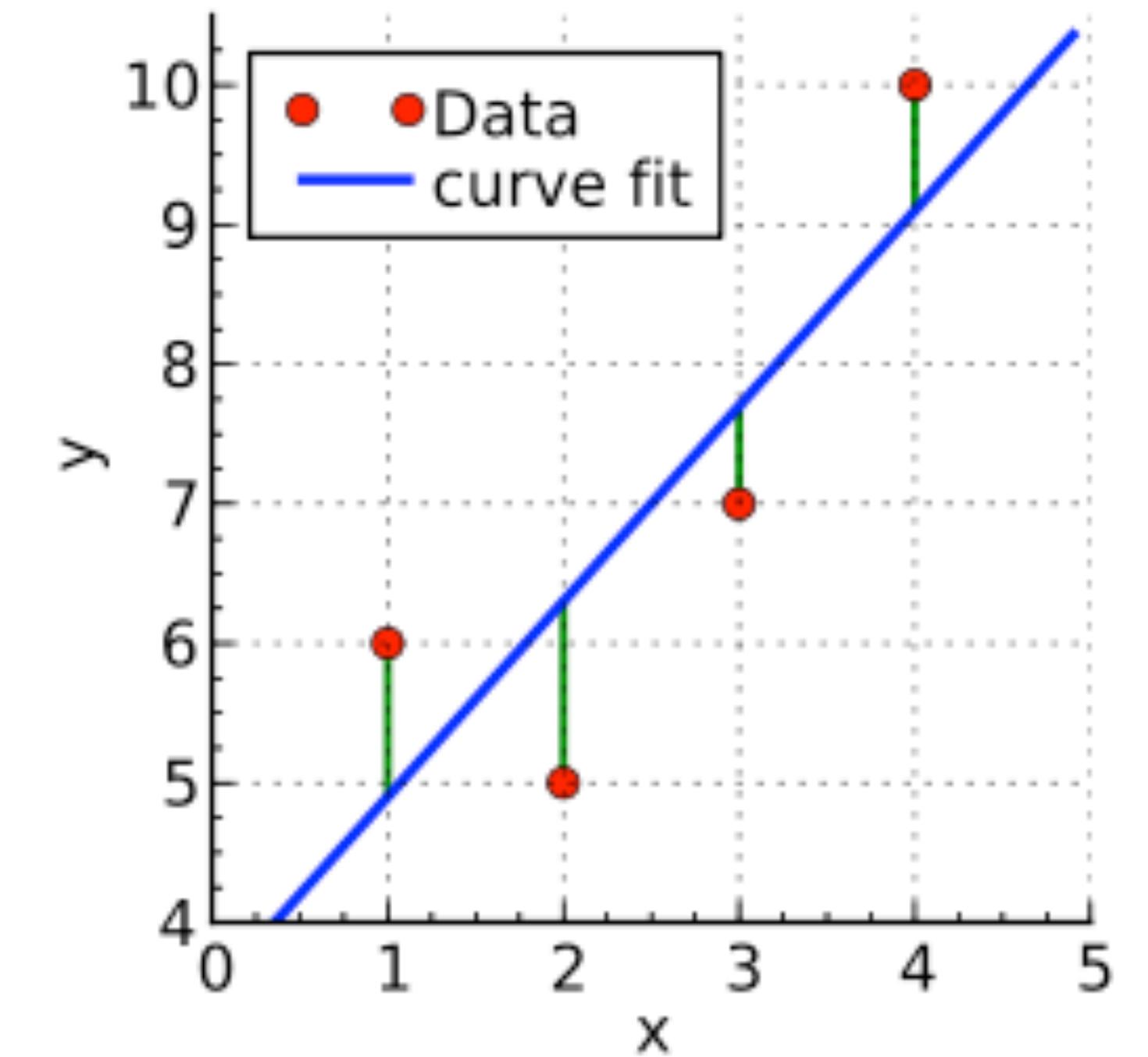
Linear regression in 1D

$$\begin{aligned}y^i &= w_0 + w_1 x_1^i + \epsilon^i && w_0 \text{ bias} \\&= w_0 x_0^i + w_1 x_1^i + \epsilon^i, & x_0^i = 1, & \forall i \\&= \mathbf{w}^T \mathbf{x}^i + \boxed{\epsilon^i} && \text{noise}\end{aligned}$$



Sum of Square Errors (MSE without the mean)

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

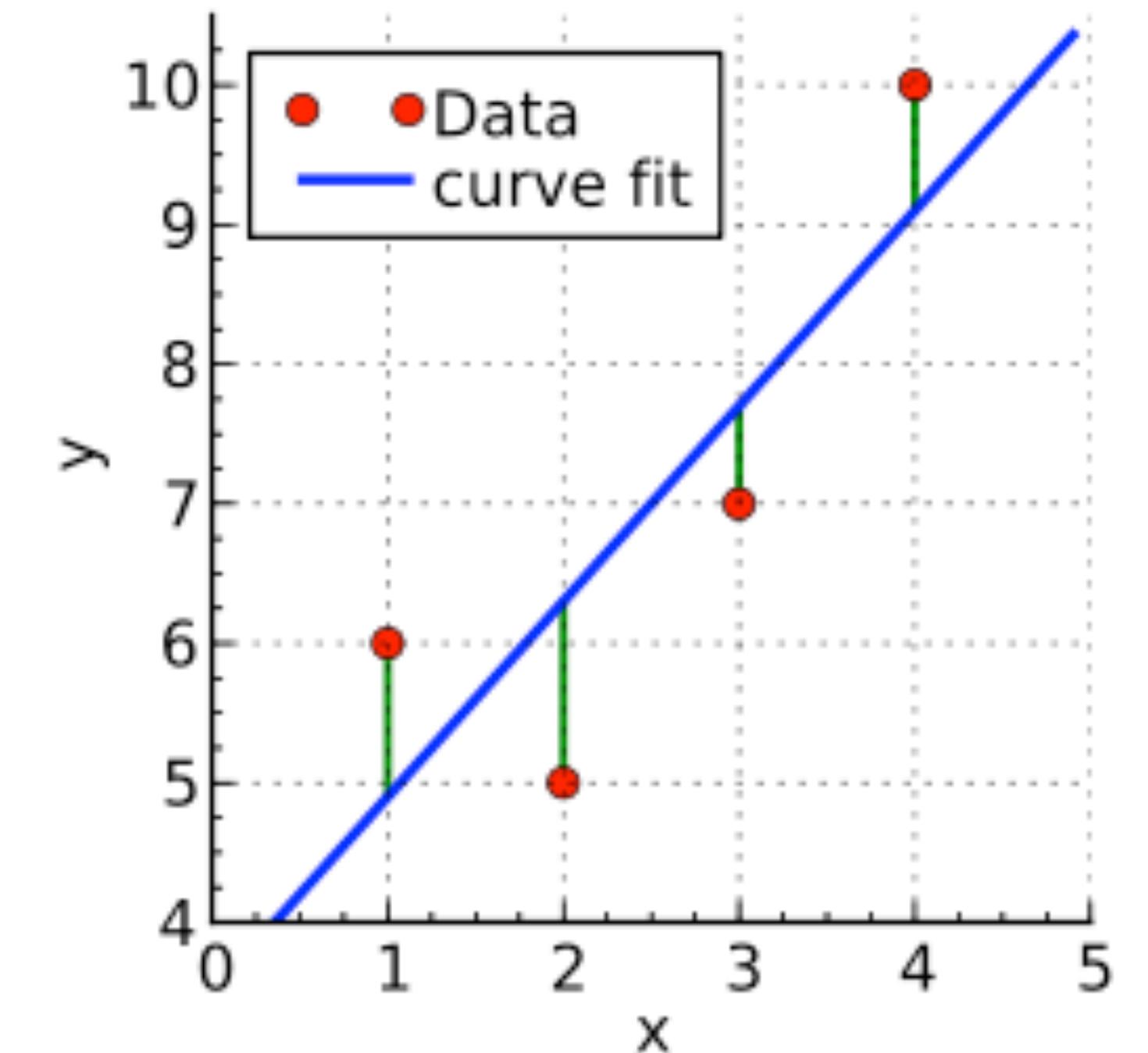


Sum of Square Errors (MSE without the mean)

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

Loss function: sum of squared errors

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



Sum of Square Errors (MSE without the mean)

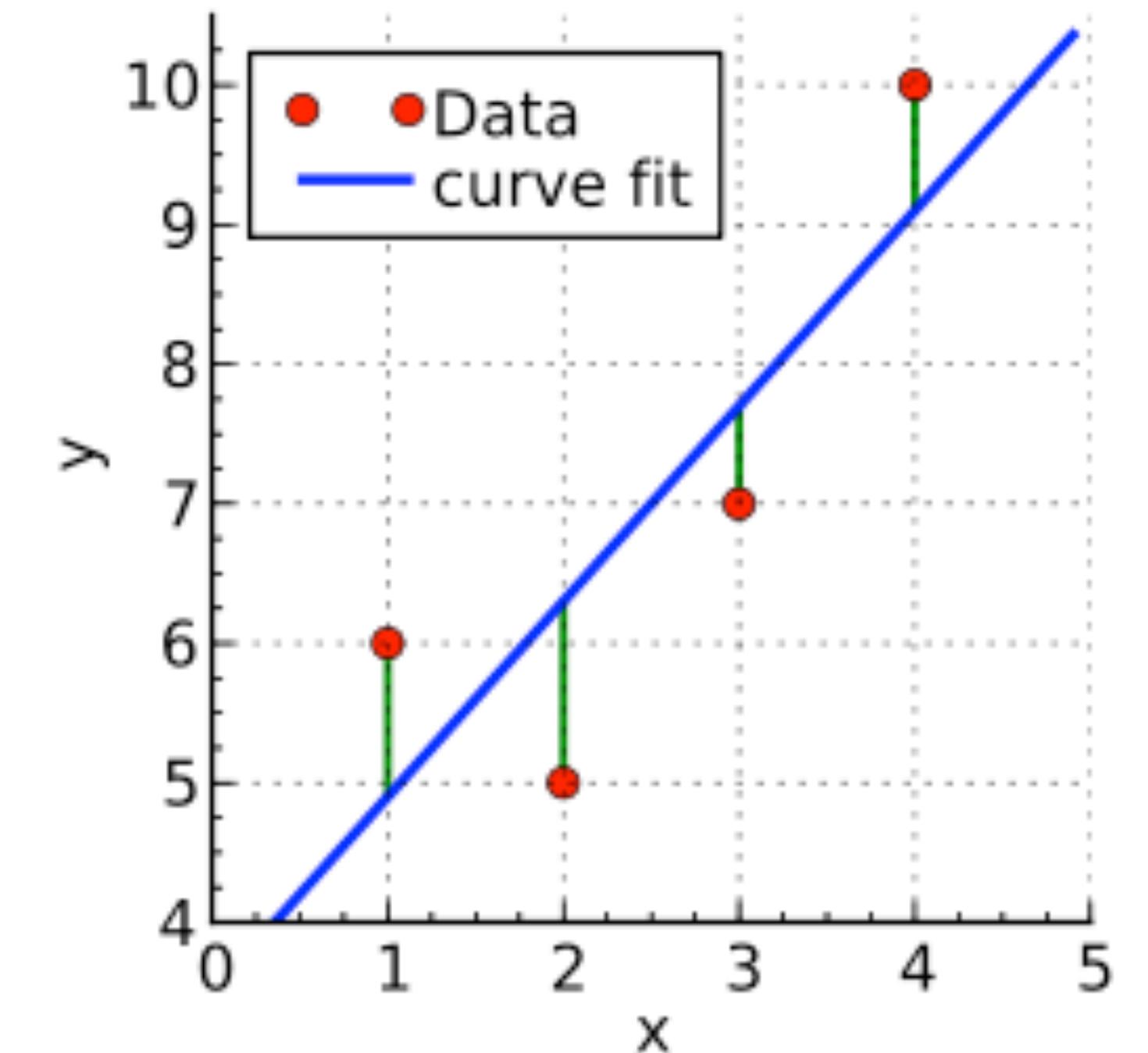
$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

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

In two variables:

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



Sum of Square Errors (MSE without the mean)

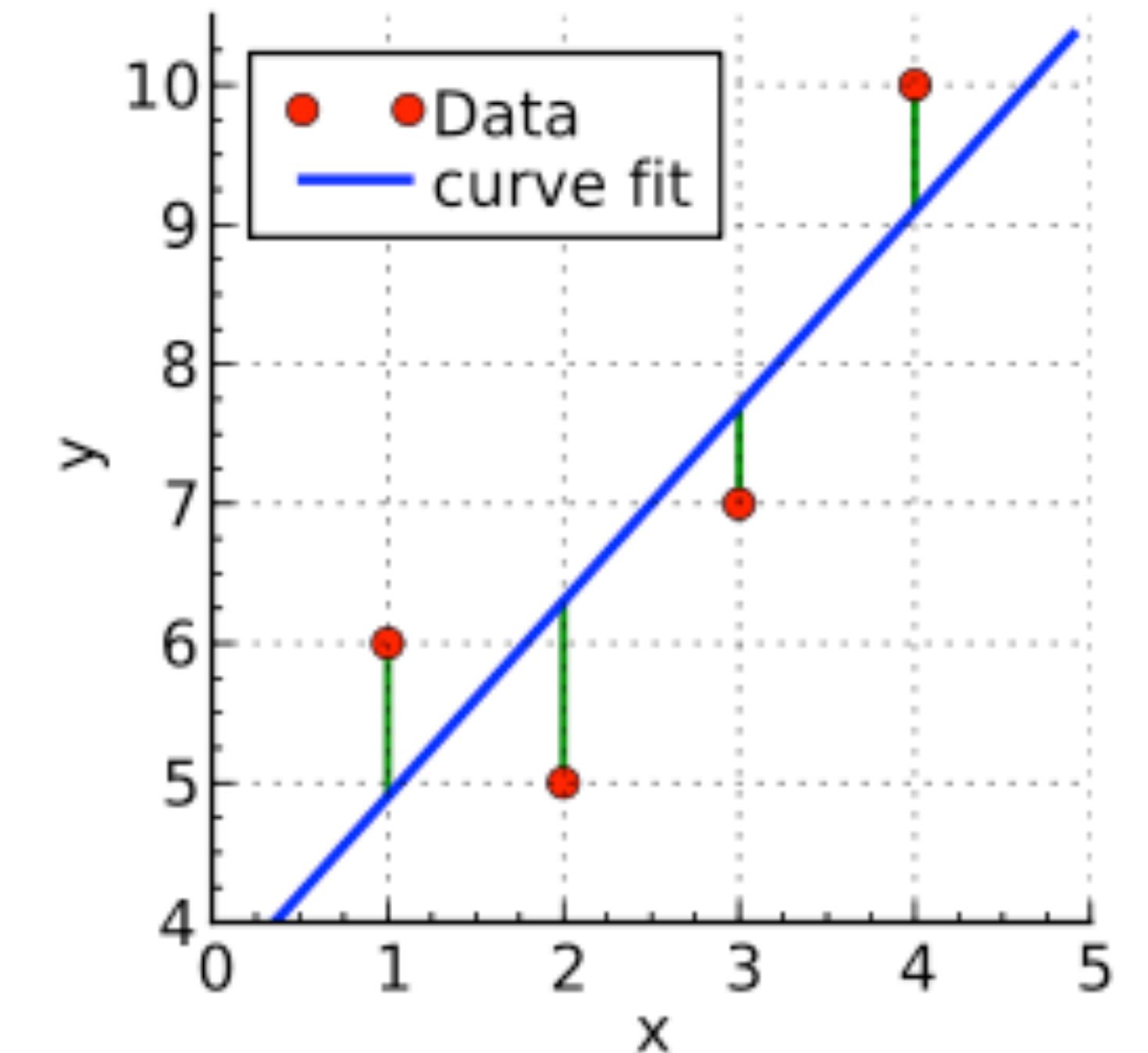
$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

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

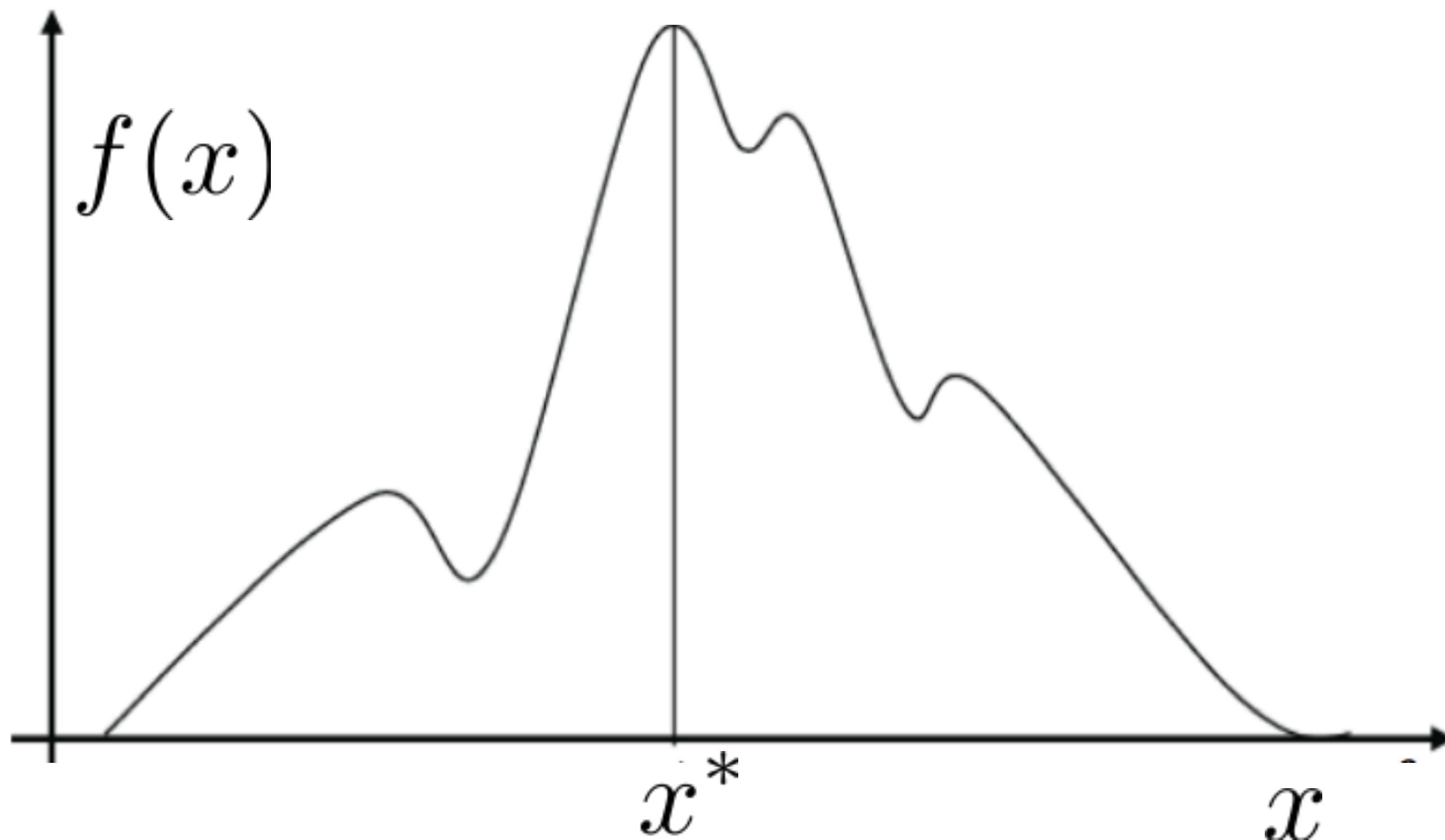
In two variables:

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

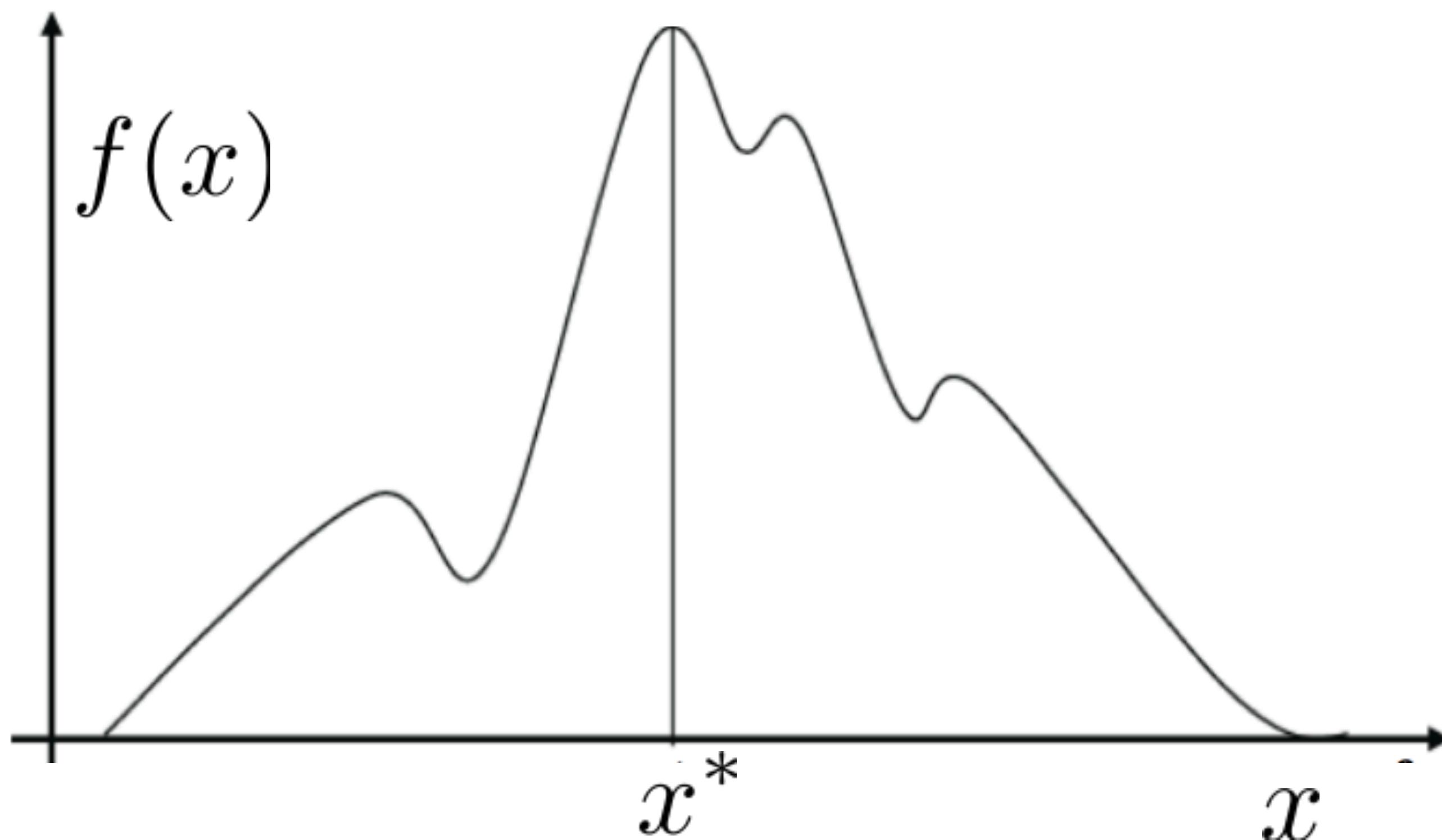


Question: what is the best (or least bad) value of w?

Calculus 101

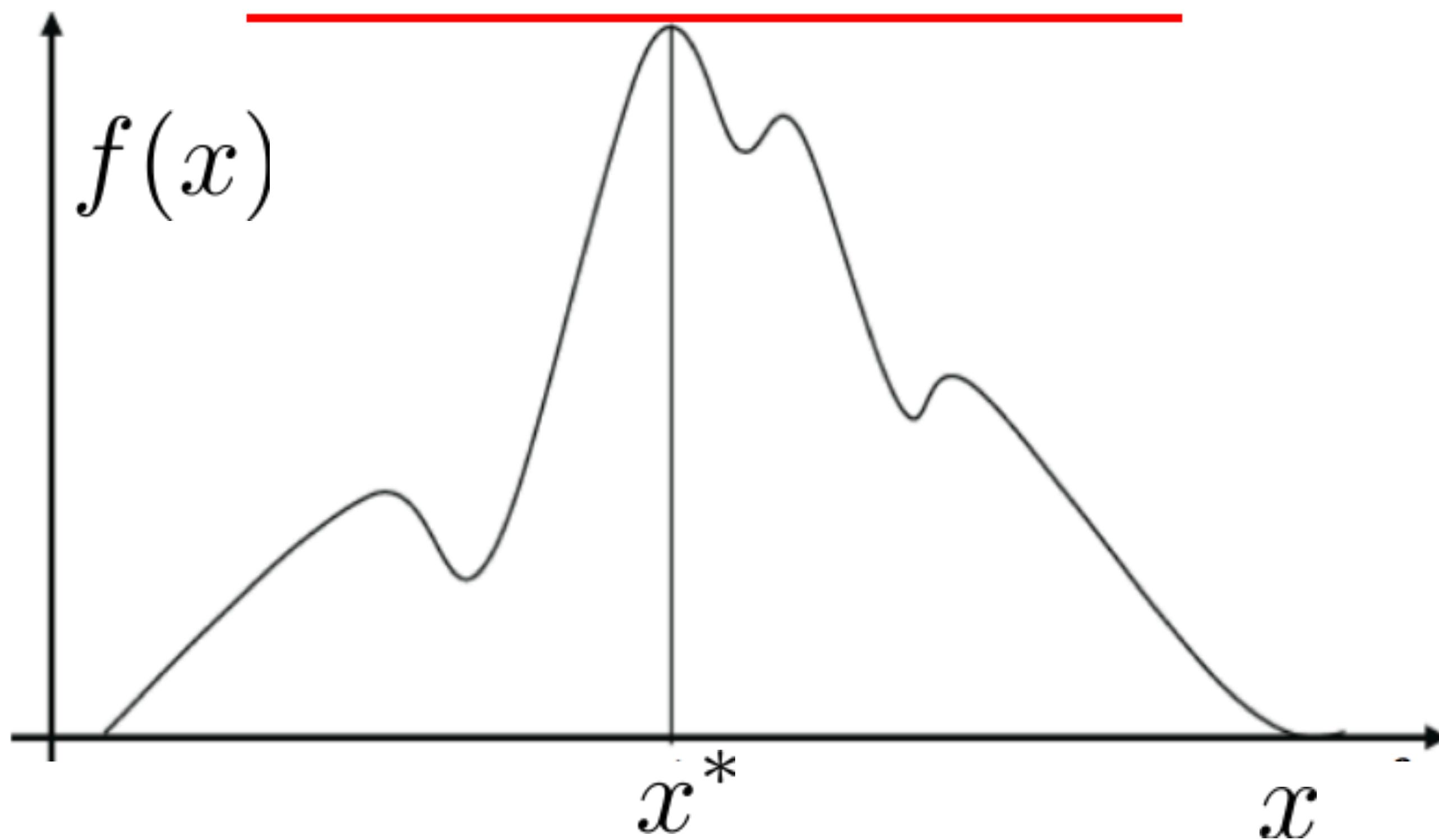


Calculus 101



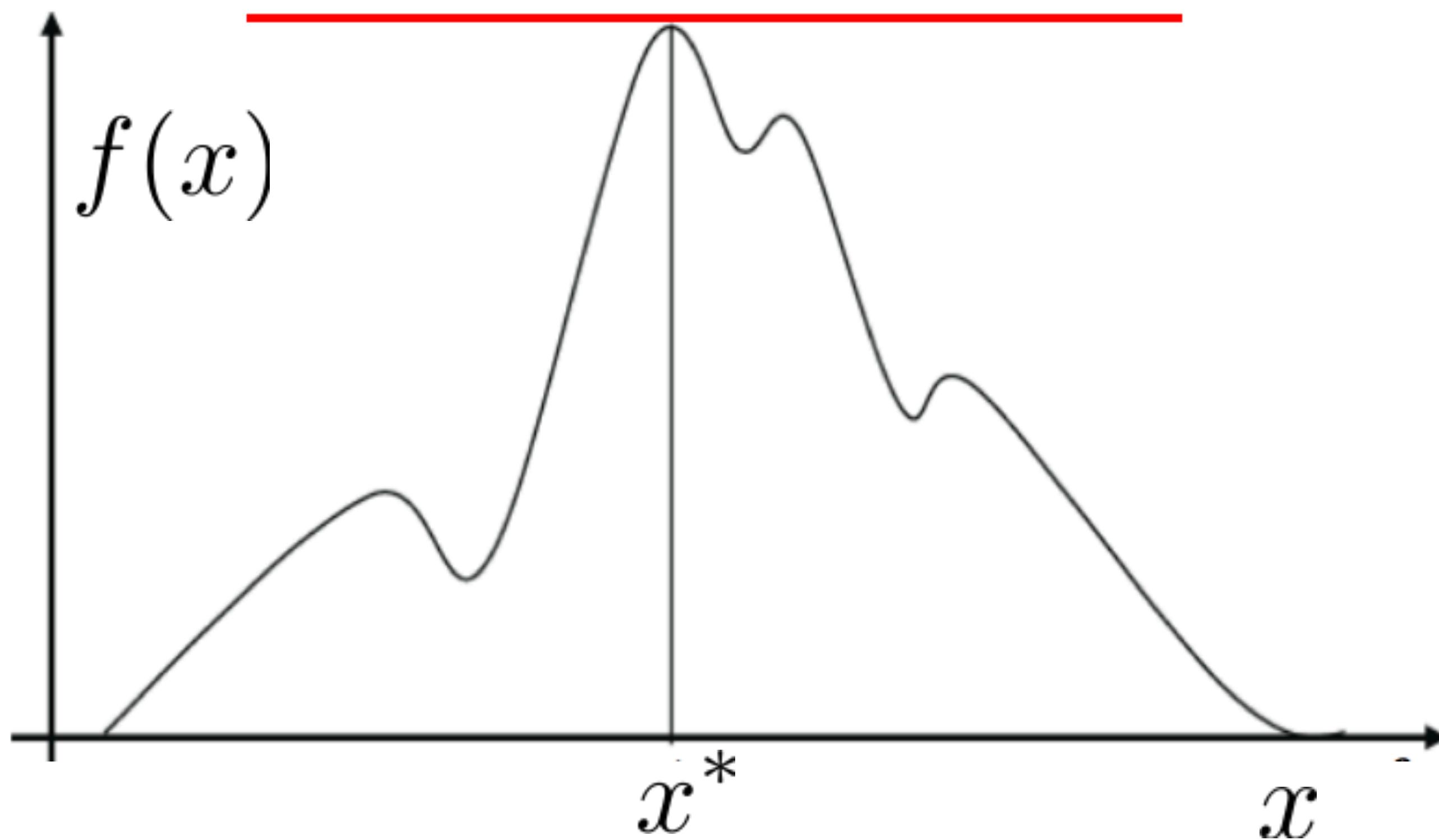
$$x^* = \operatorname{argmax}_x f(x)$$

Local Extrema Condition



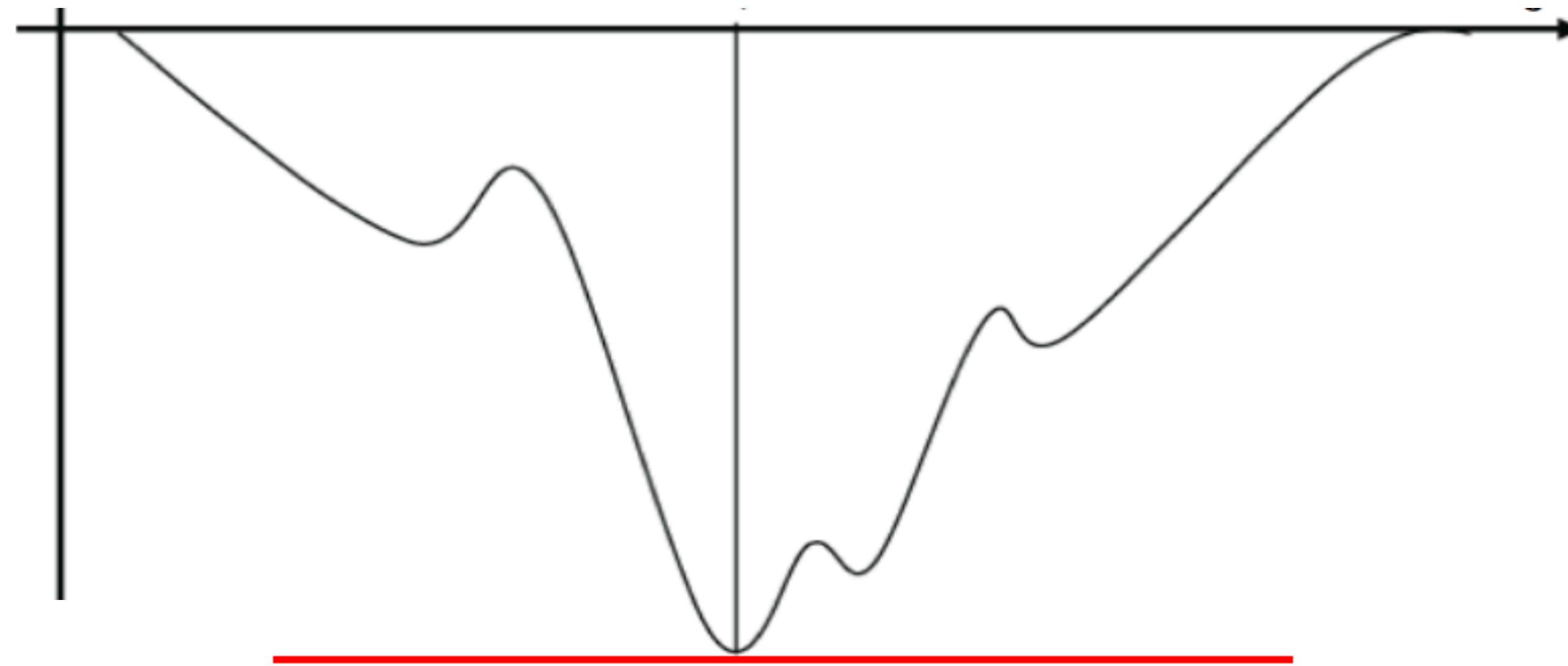
$$x^* = \operatorname{argmax}_x f(x)$$

Local Extrema Condition



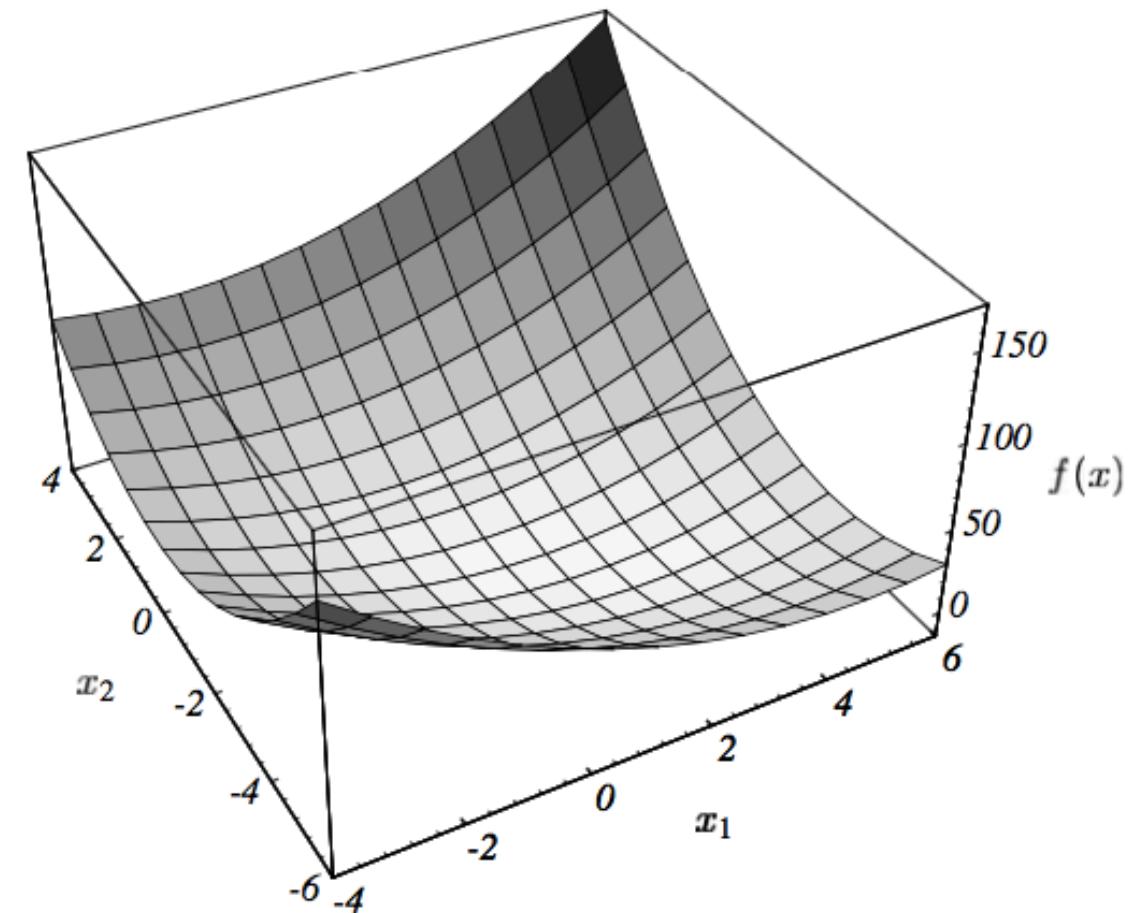
$$x^* = \operatorname{argmax}_x f(x) \rightarrow f'(x^*) = 0$$

Local Extrema Condition



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

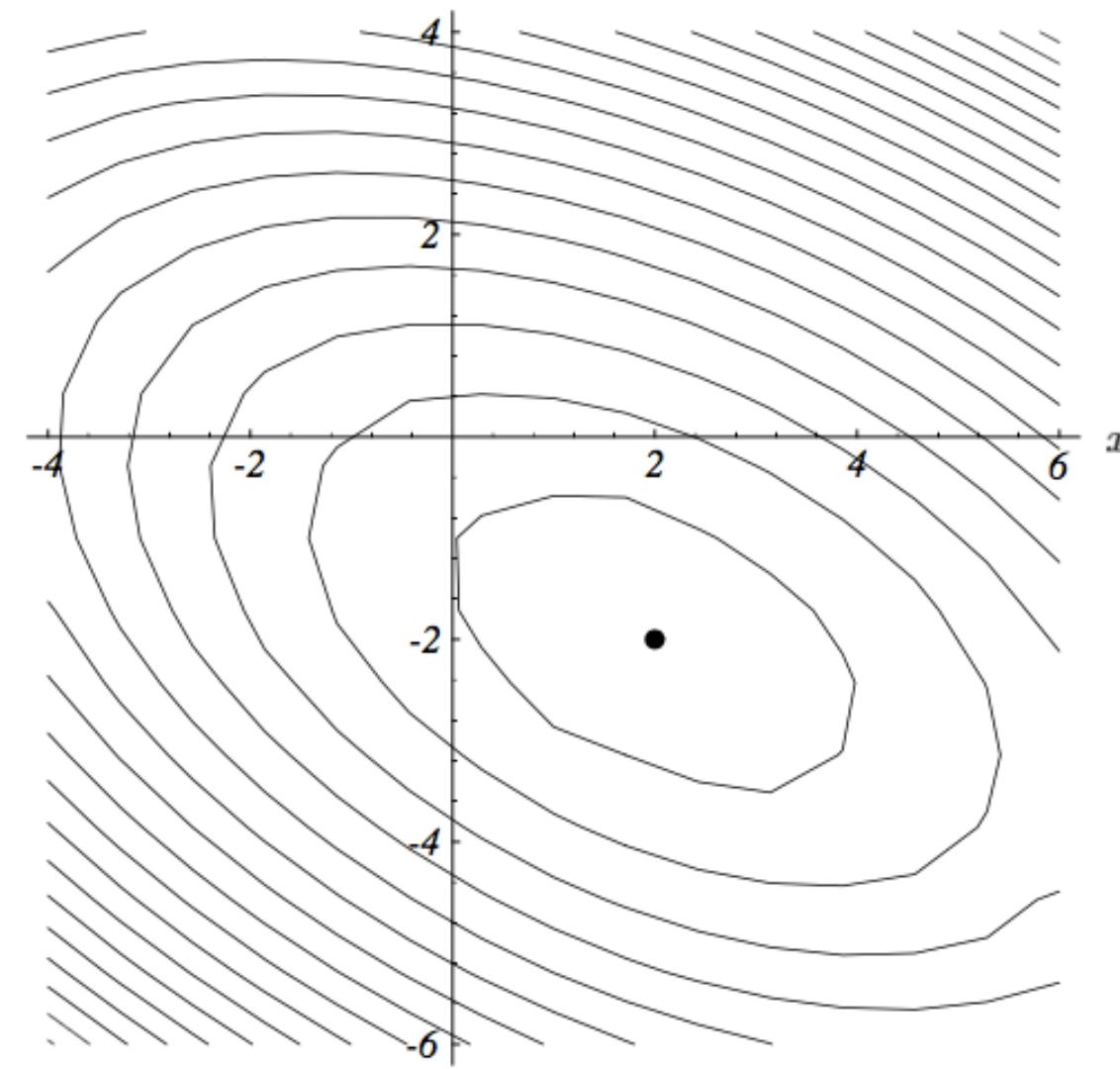
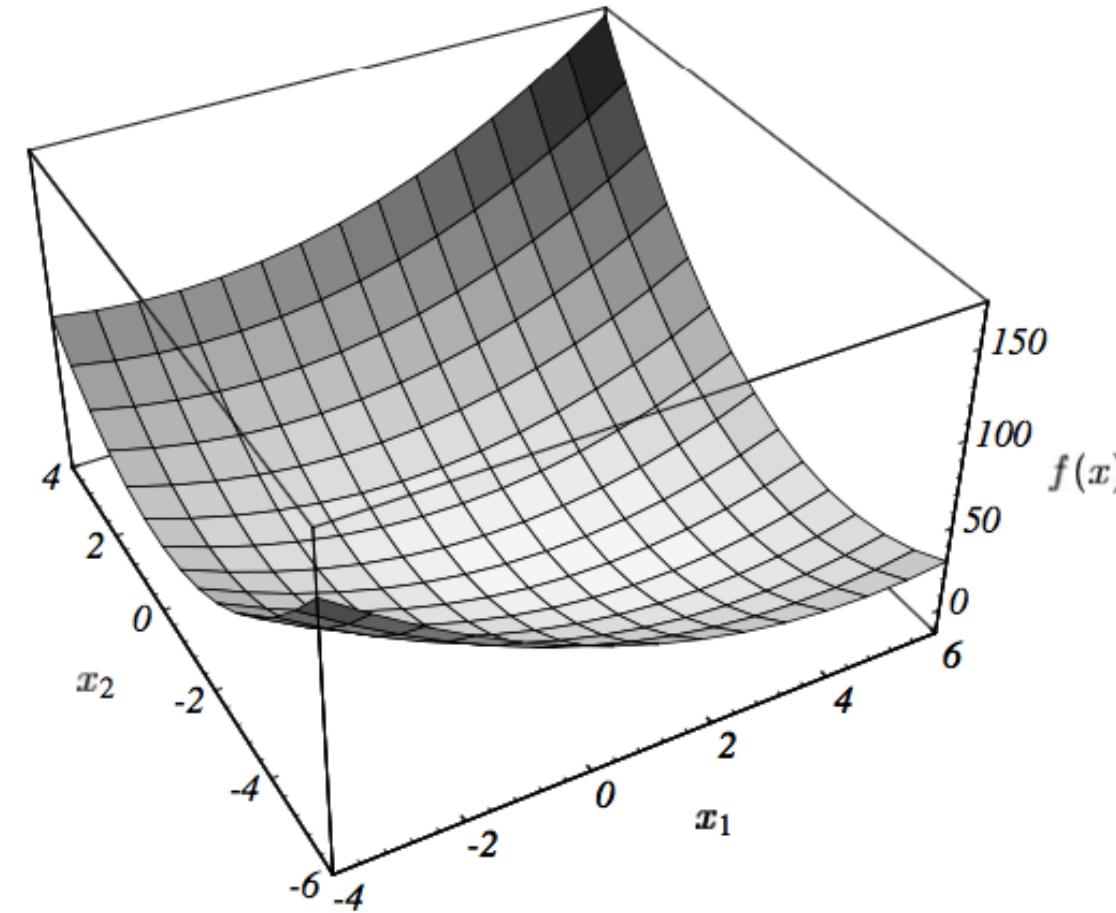
Vector Calculus 101



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

2D function graph

Vector Calculus 101



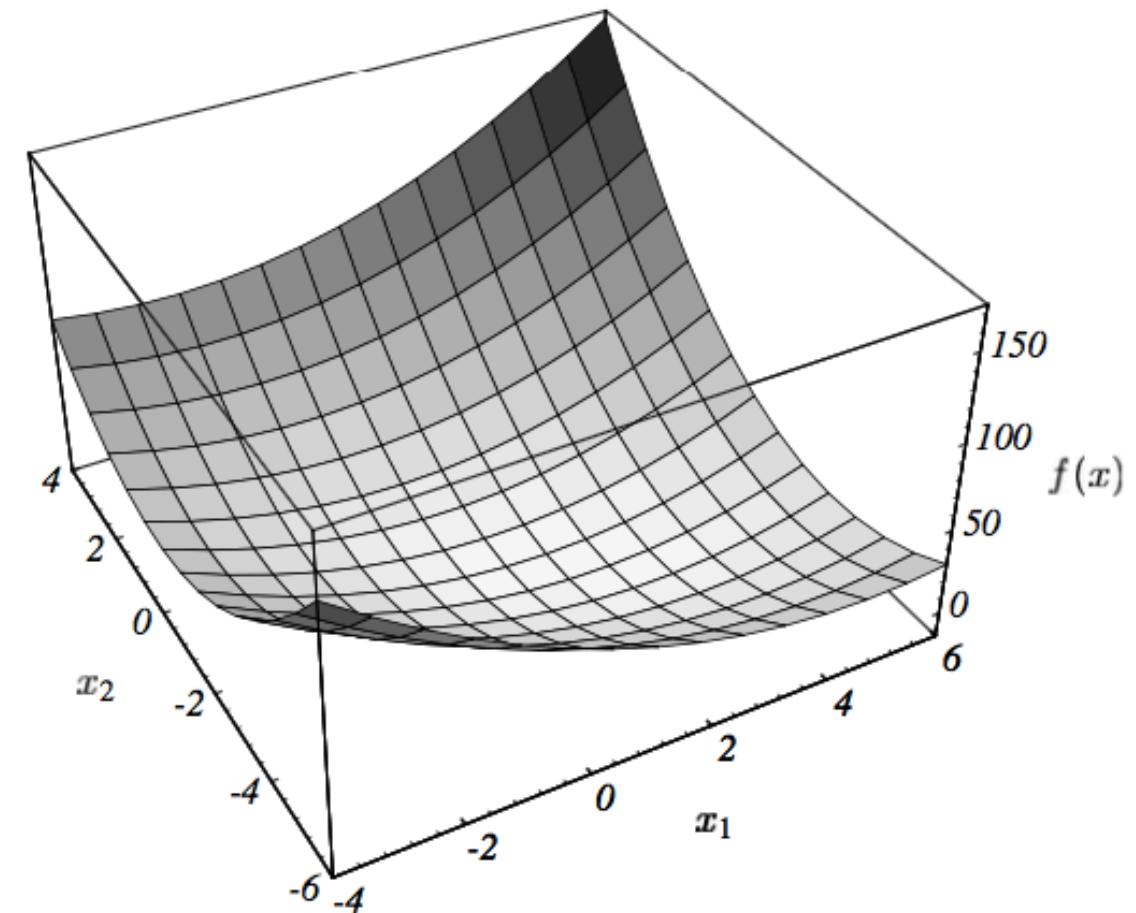
$$f(\mathbf{x})$$

2D function graph

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

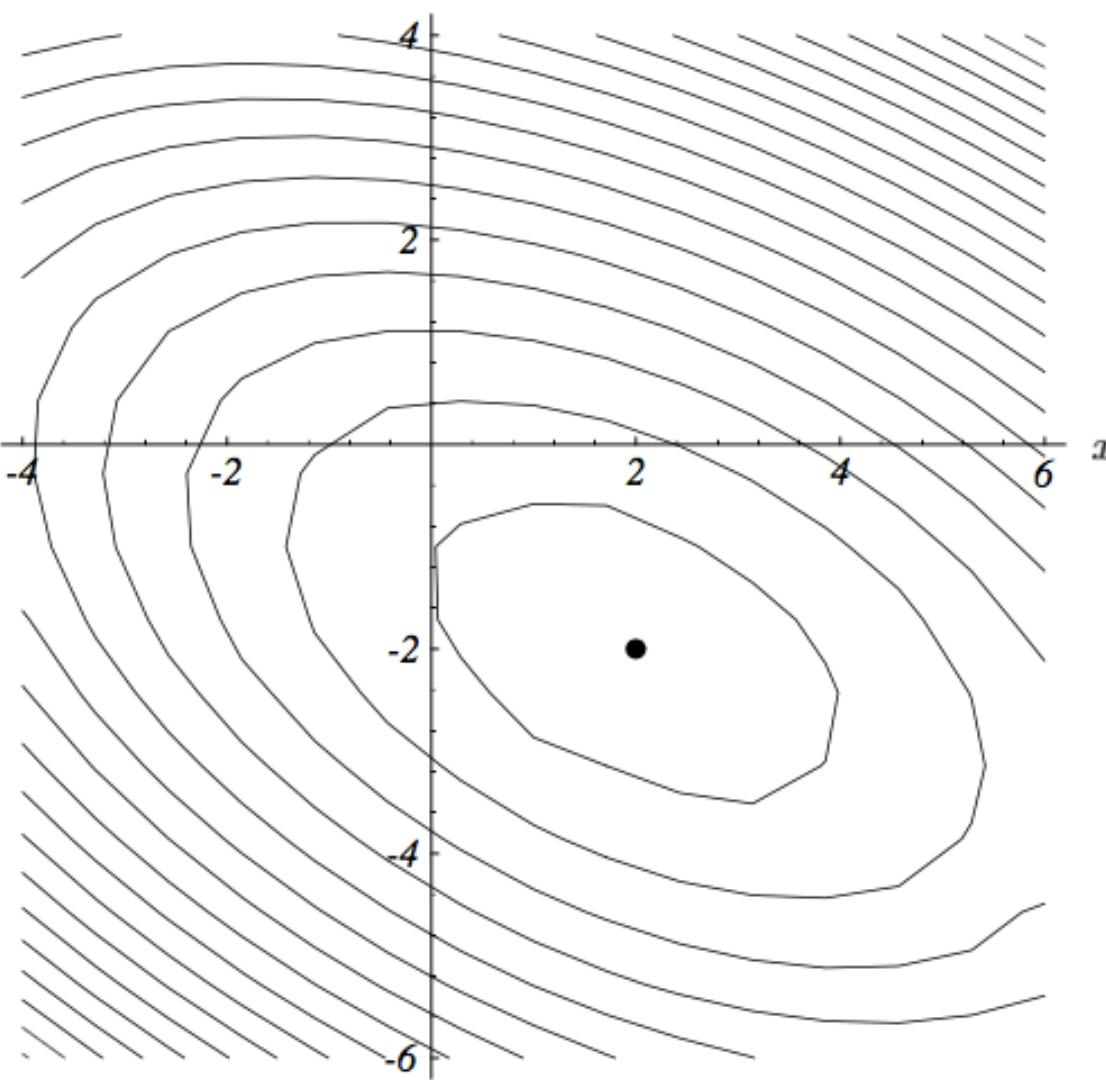
isocontours

Vector Calculus 101



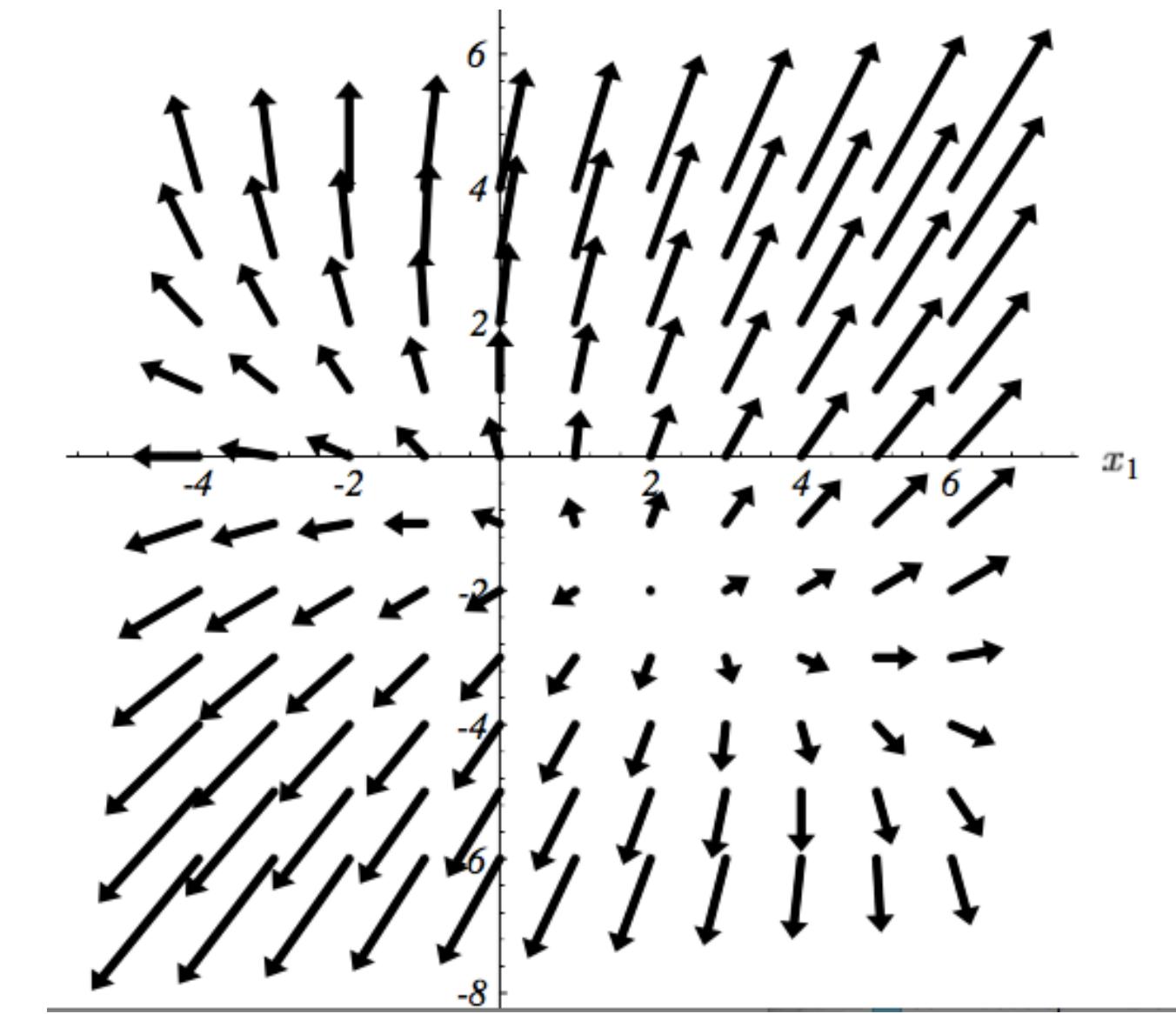
$$f(\mathbf{x})$$

2D function graph



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

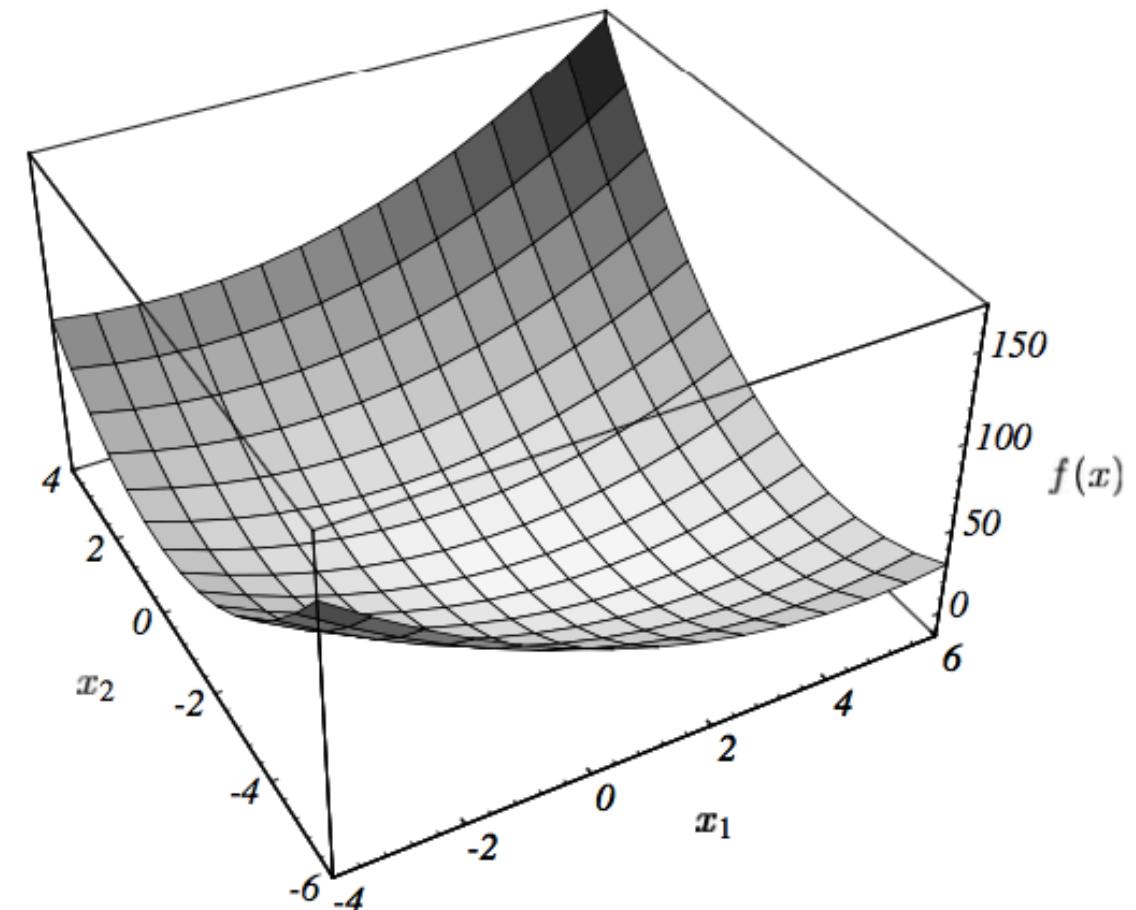
isocontours



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

gradient field

Vector Calculus 101

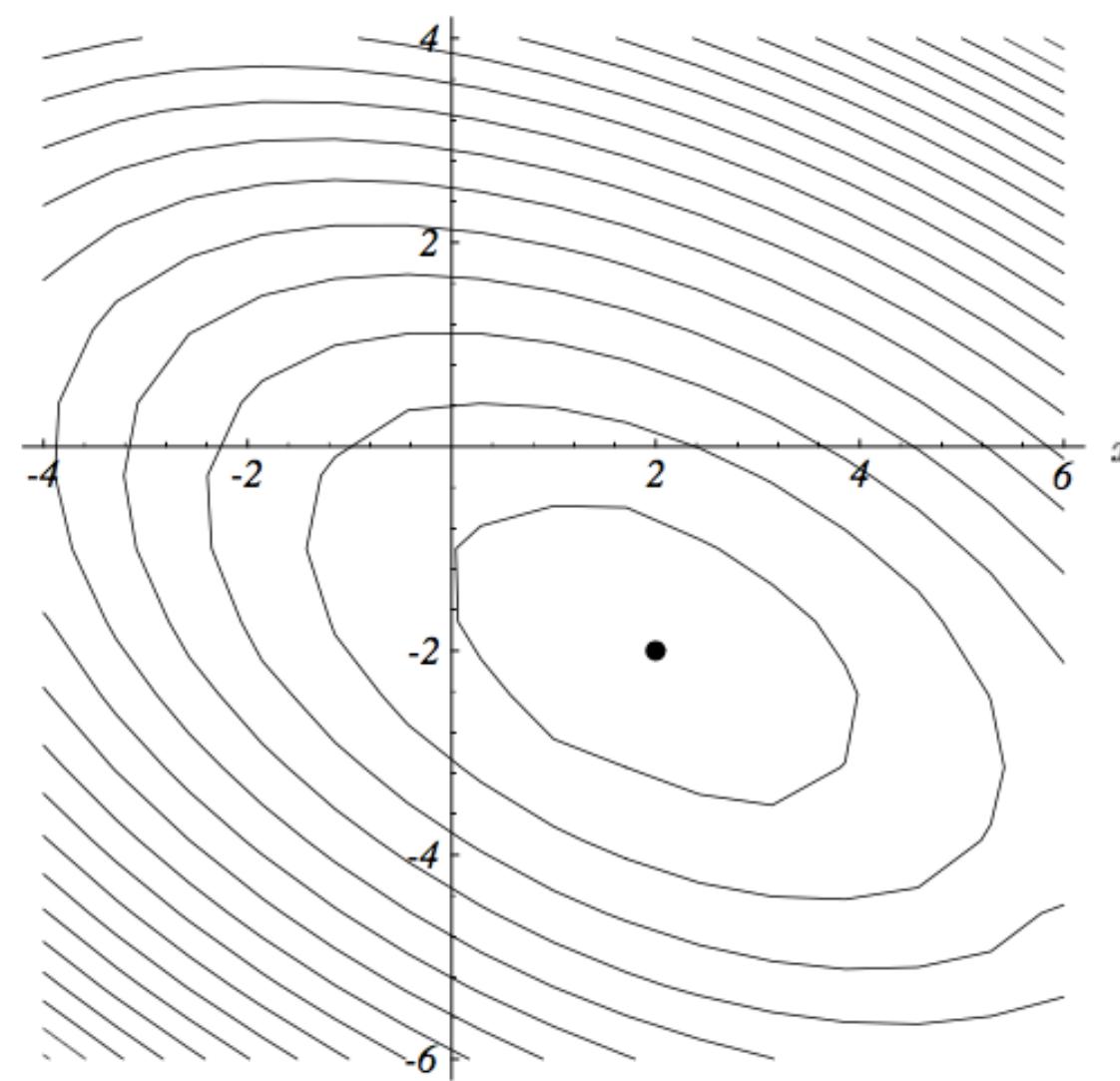


$$f(\mathbf{x})$$

2D function graph

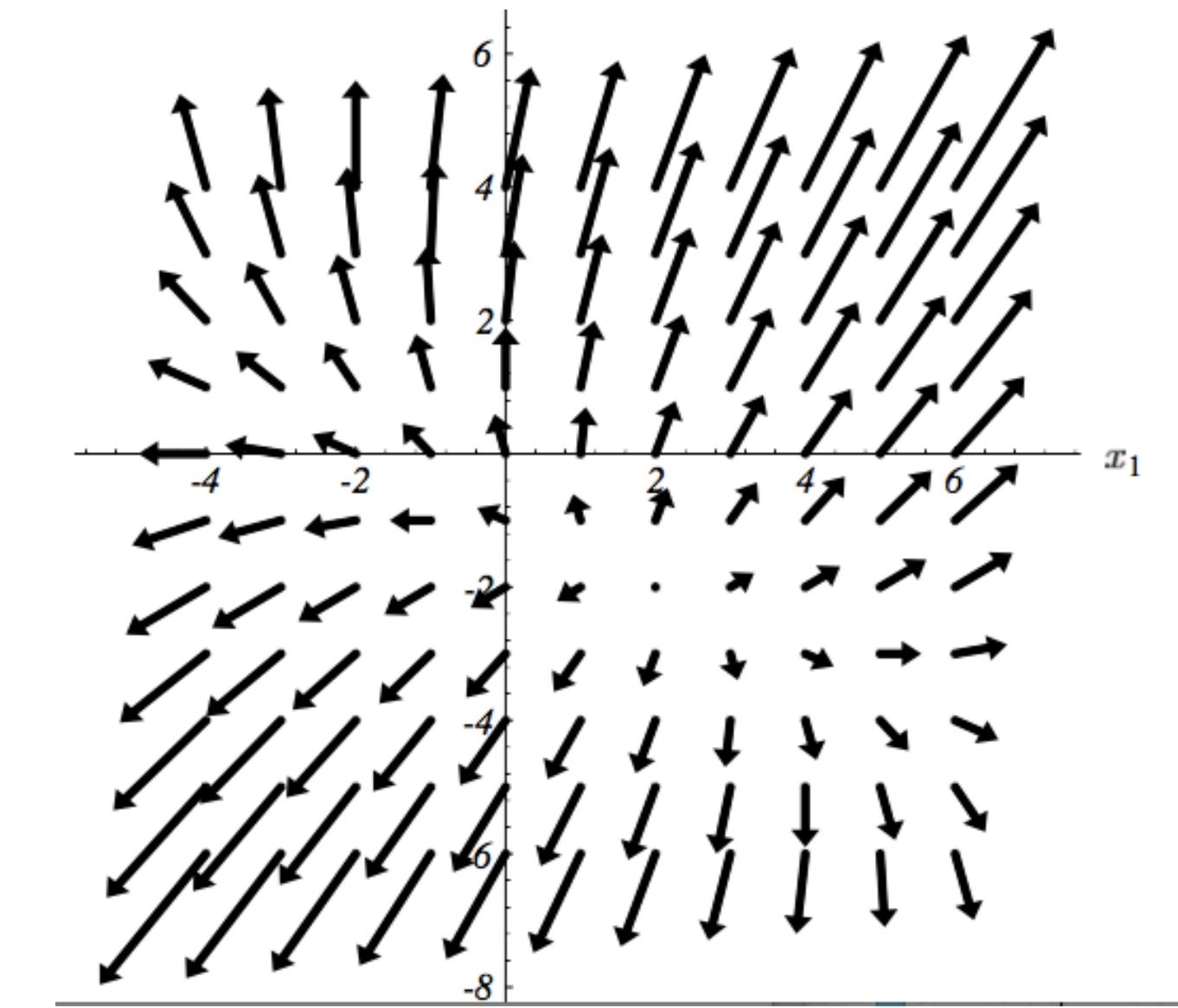


at minimum of function:



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

isocontours

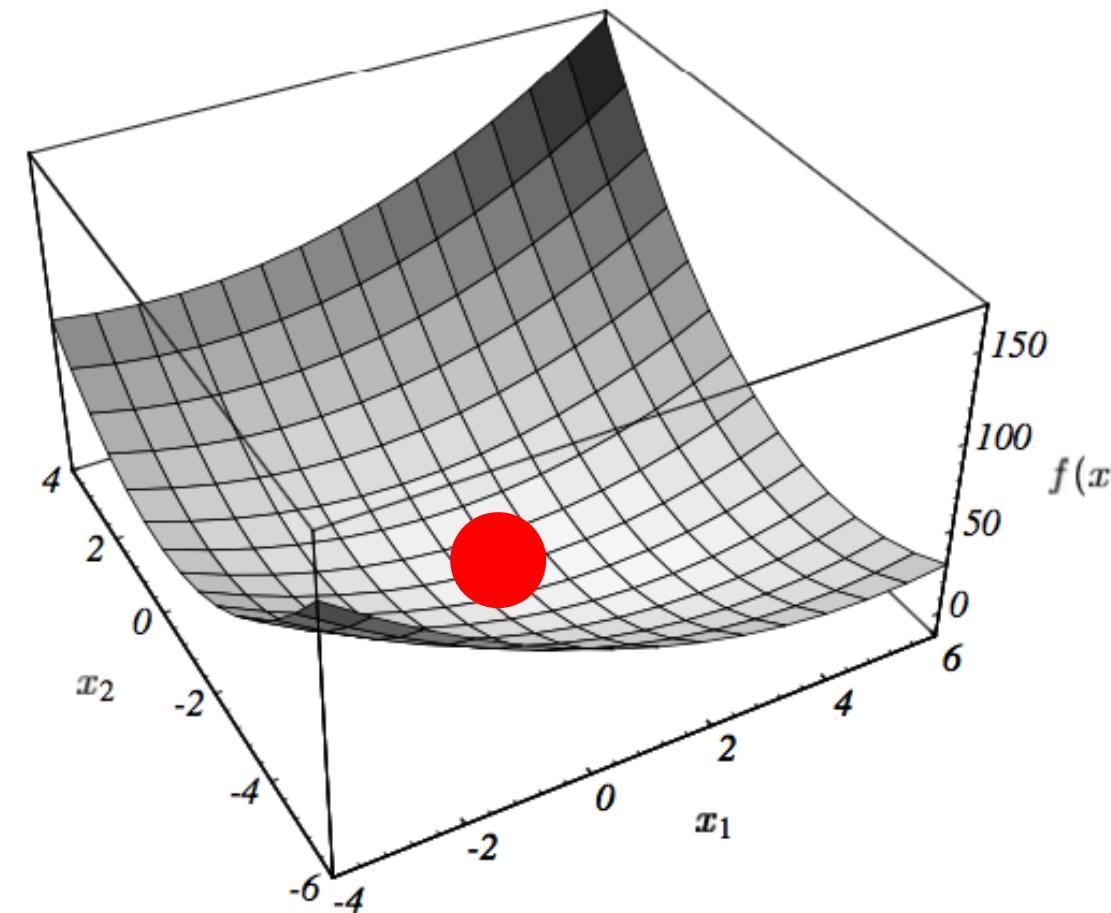


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

gradient field

$$\nabla f(\mathbf{x}) = 0$$

Vector Calculus 101

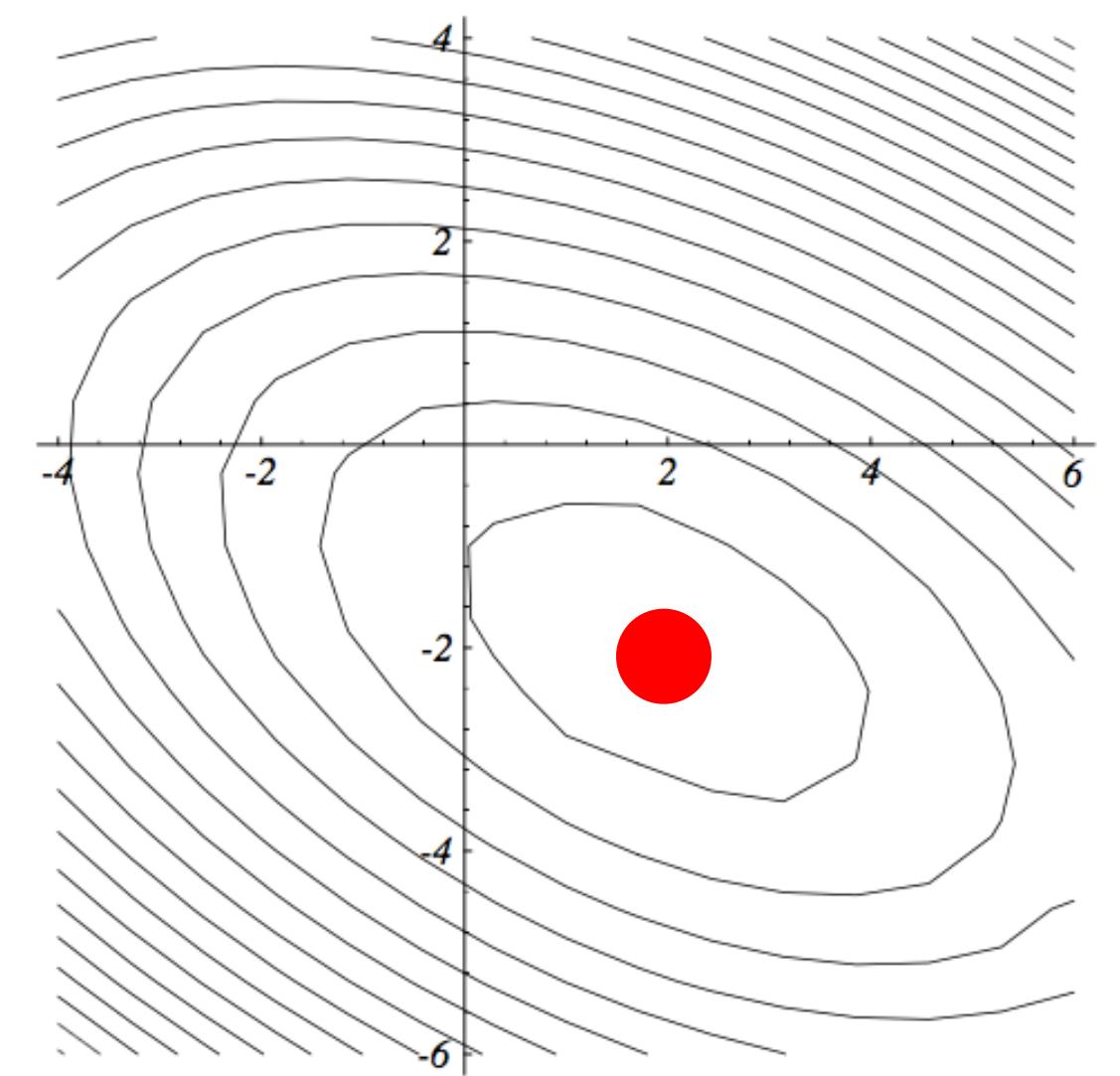


$$f(\mathbf{x})$$

2D function graph

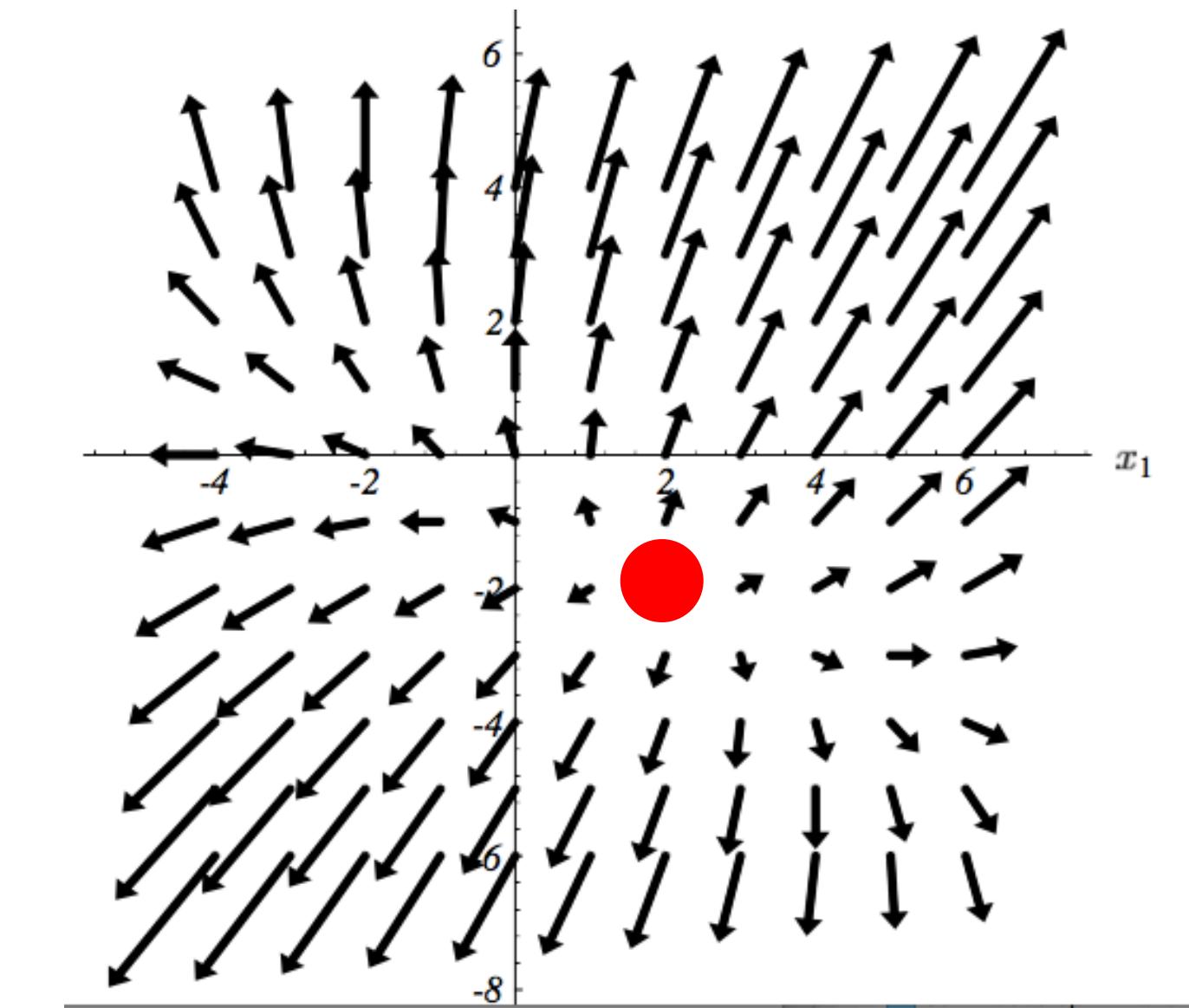


at minimum of function:



isocontours

kNN and Regression



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

gradient field

$$\nabla f(\mathbf{x}) = 0$$

Line Fitting

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

Line Fitting

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

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

Line Fitting

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

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

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

Line Fitting

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

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

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

Line Fitting

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

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

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

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

Line Fitting

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

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

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

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

Line Fitting

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

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

$$\begin{aligned}\frac{\partial L(w_0, w_1)}{\partial w_0} &= \sum_{i=1}^N \frac{\partial [z^i]^2}{\partial z^i} \frac{\partial z^i}{\partial w_0} &= \sum_{i=1}^N (2z^i)(-x_0^i) \\ &= -2 \sum_{i=1}^N (y^i - (w_0 x_0^i + w_1 x_1^i)) x_0^i\end{aligned}$$

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

Line Fitting (continued)

Line Fitting (continued)

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

Line Fitting (continued)

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

Line Fitting (continued)

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

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

Line Fitting (continued)

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

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

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

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)

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

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

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

Line Fitting (continued)

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

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

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

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

Line Fitting (continued)

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

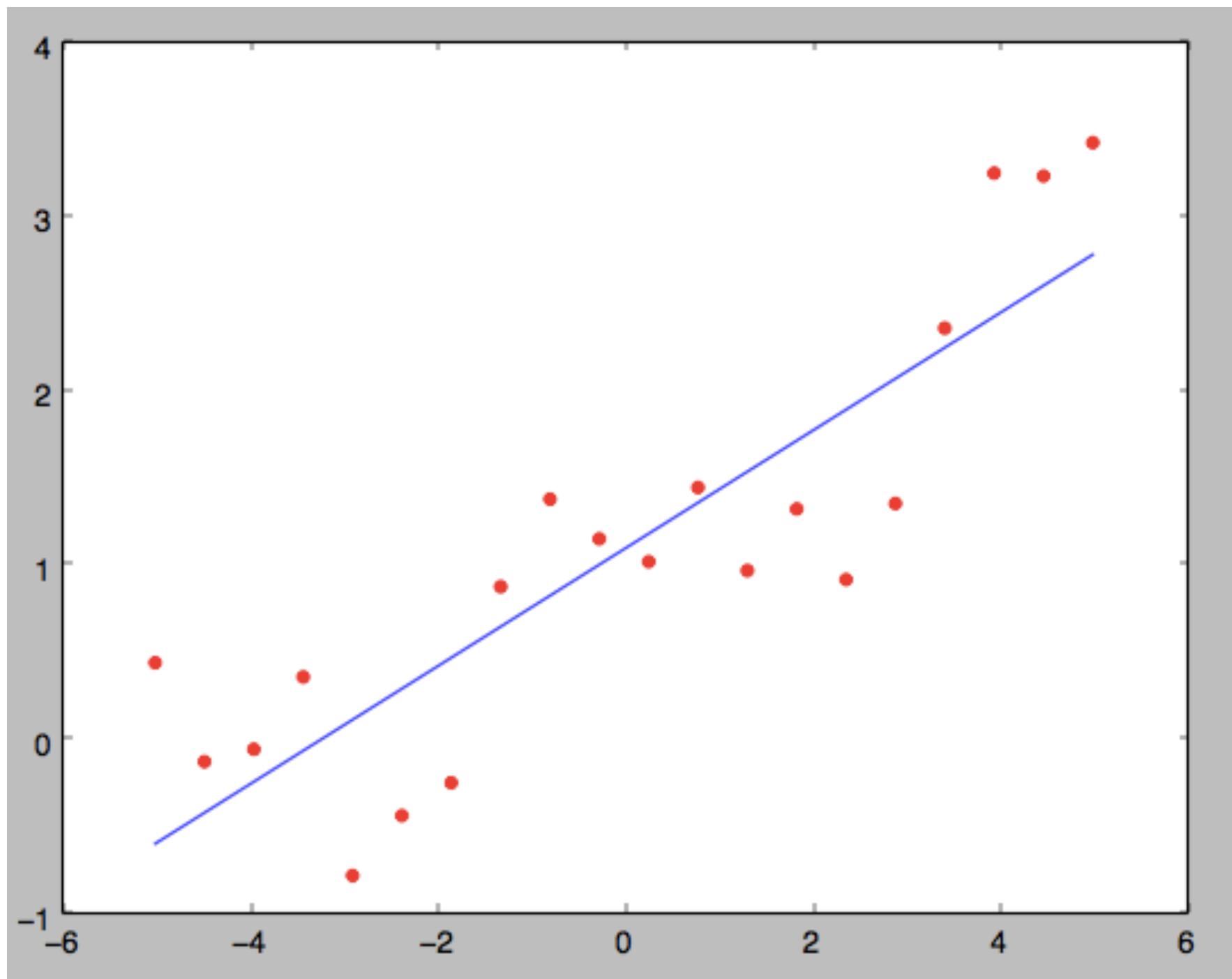
$$\mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X} \mathbf{w}$$

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

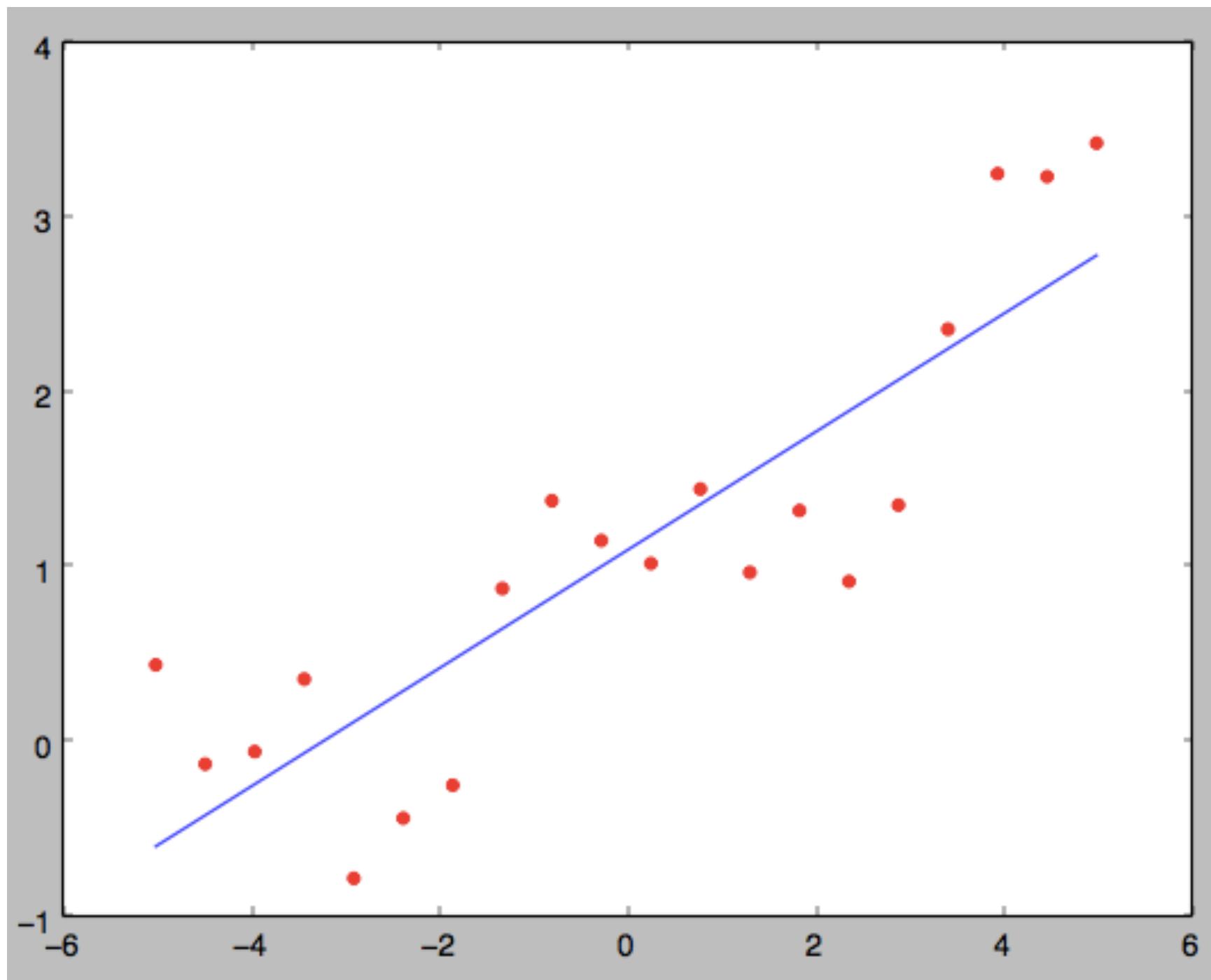
Normal Equation

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

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)] )

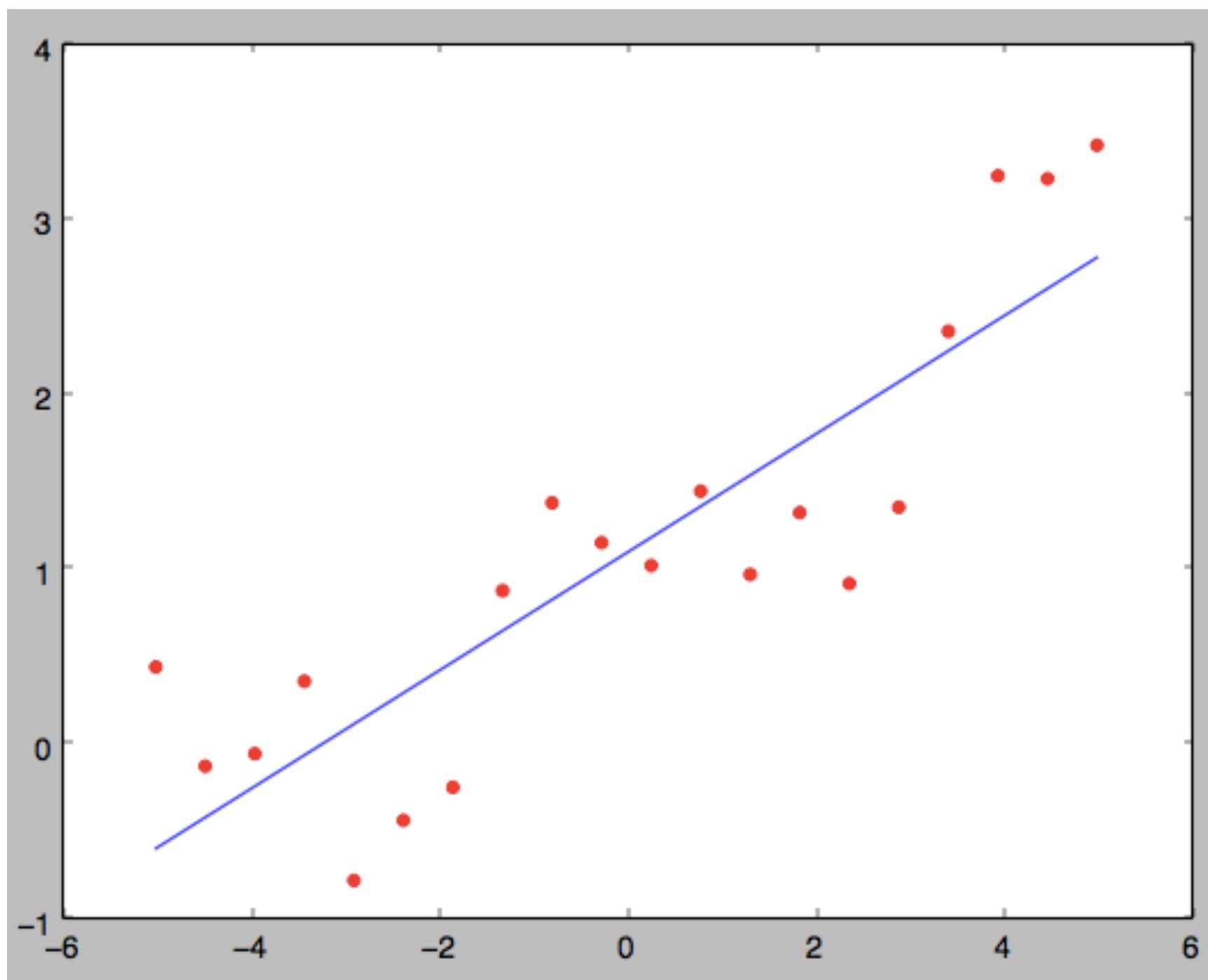
# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# 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



```
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)] )

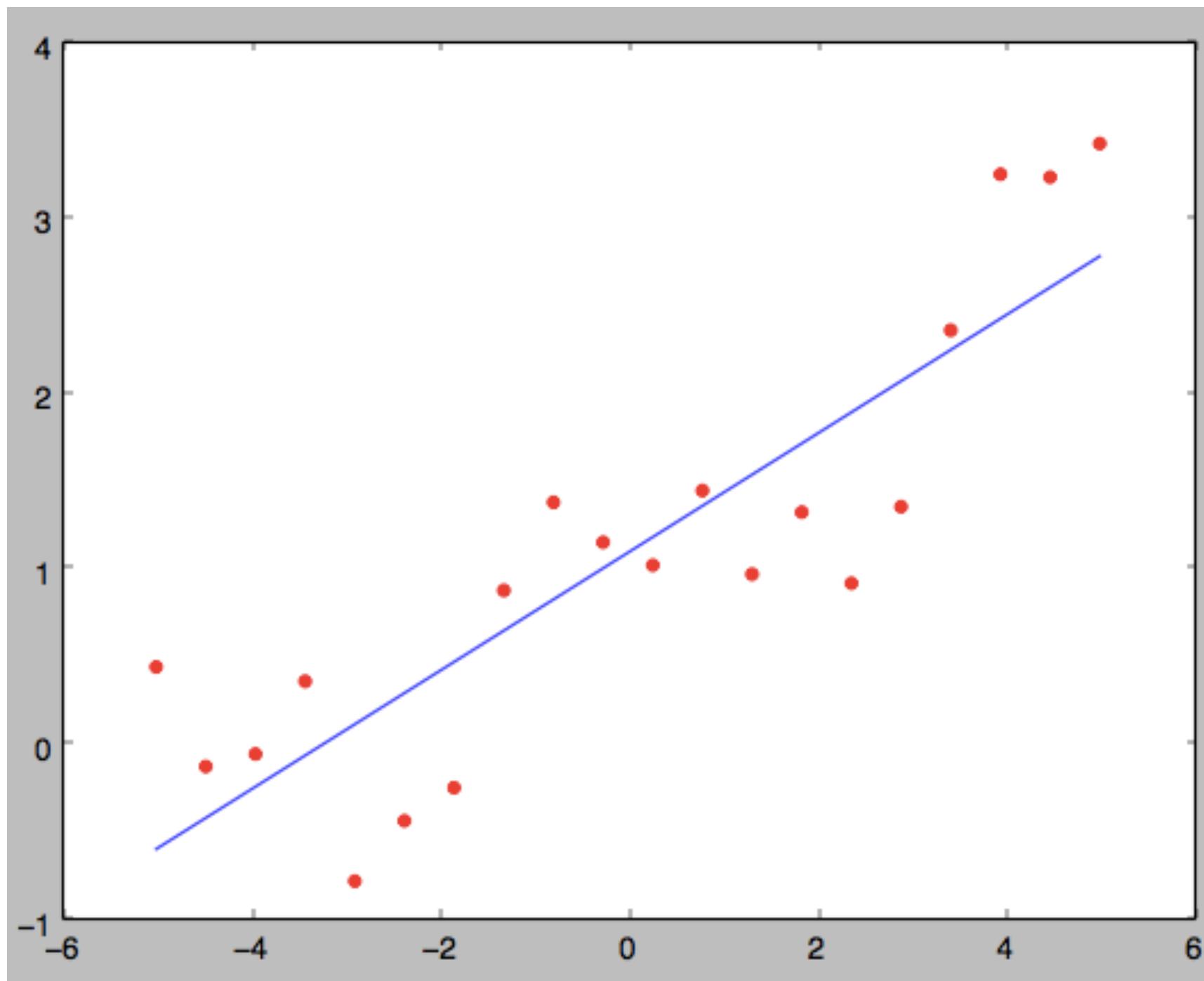
# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# 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)

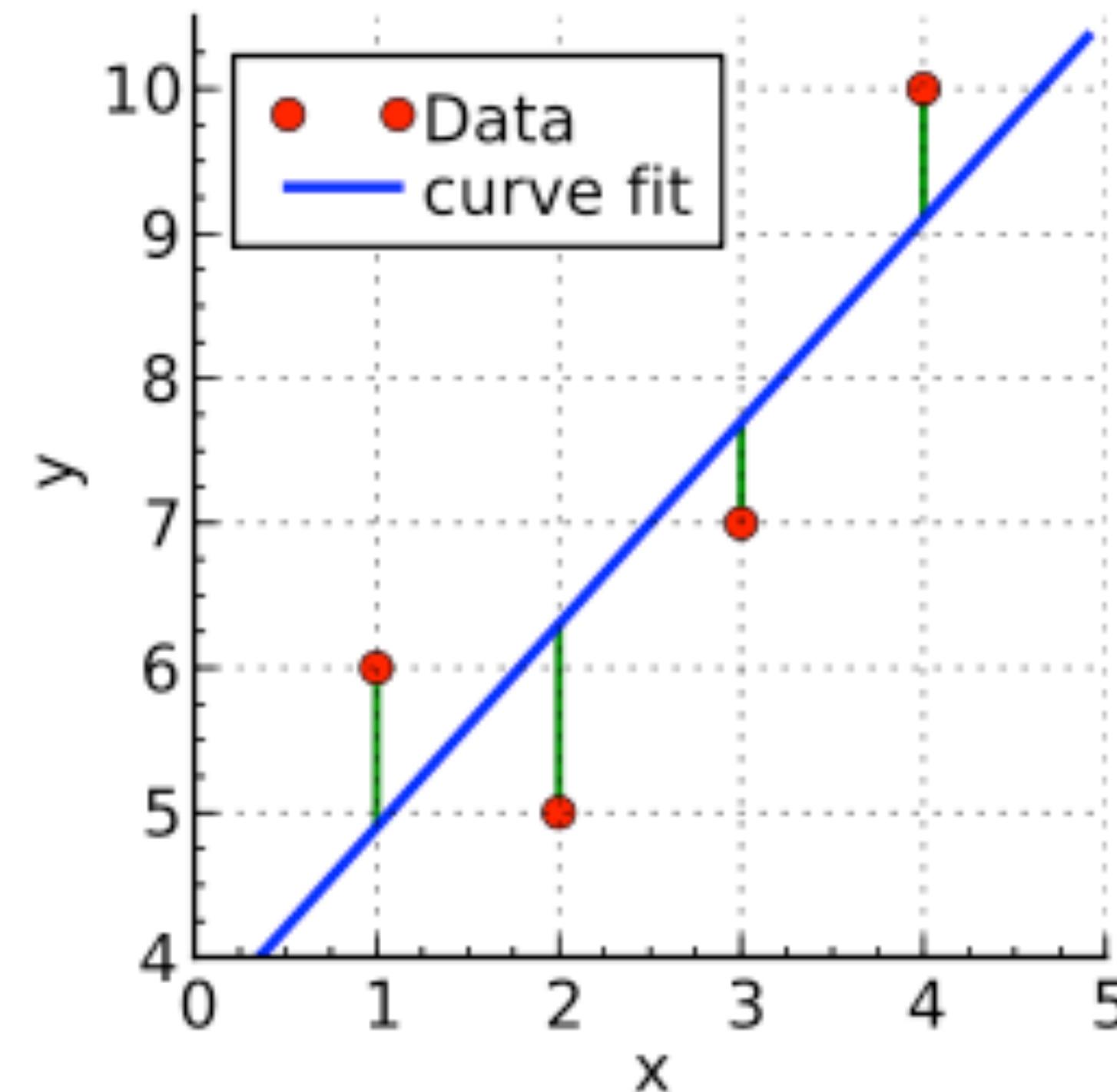
# For ridge regression, use regularizer
#weight = 0.01
#w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)

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

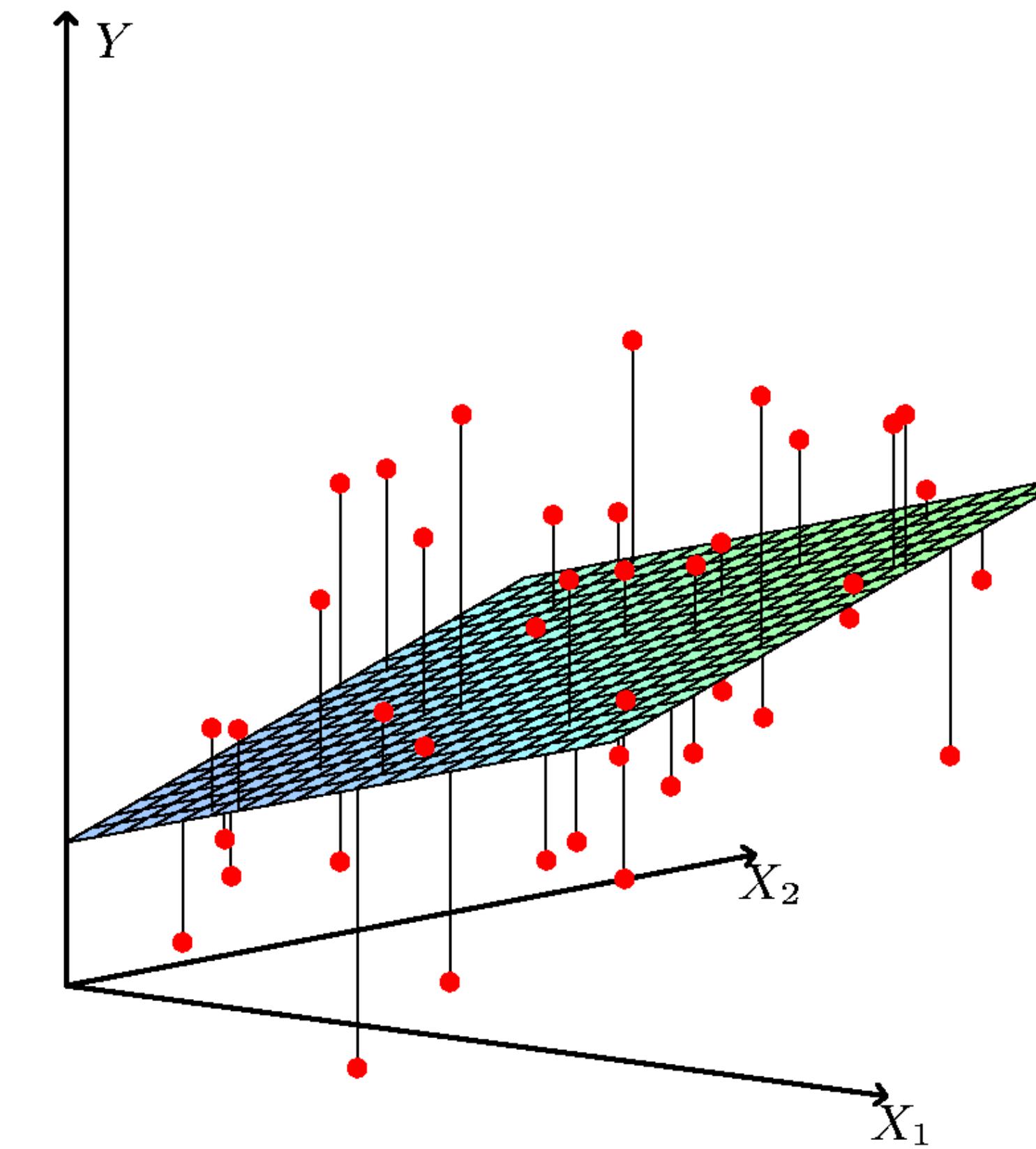
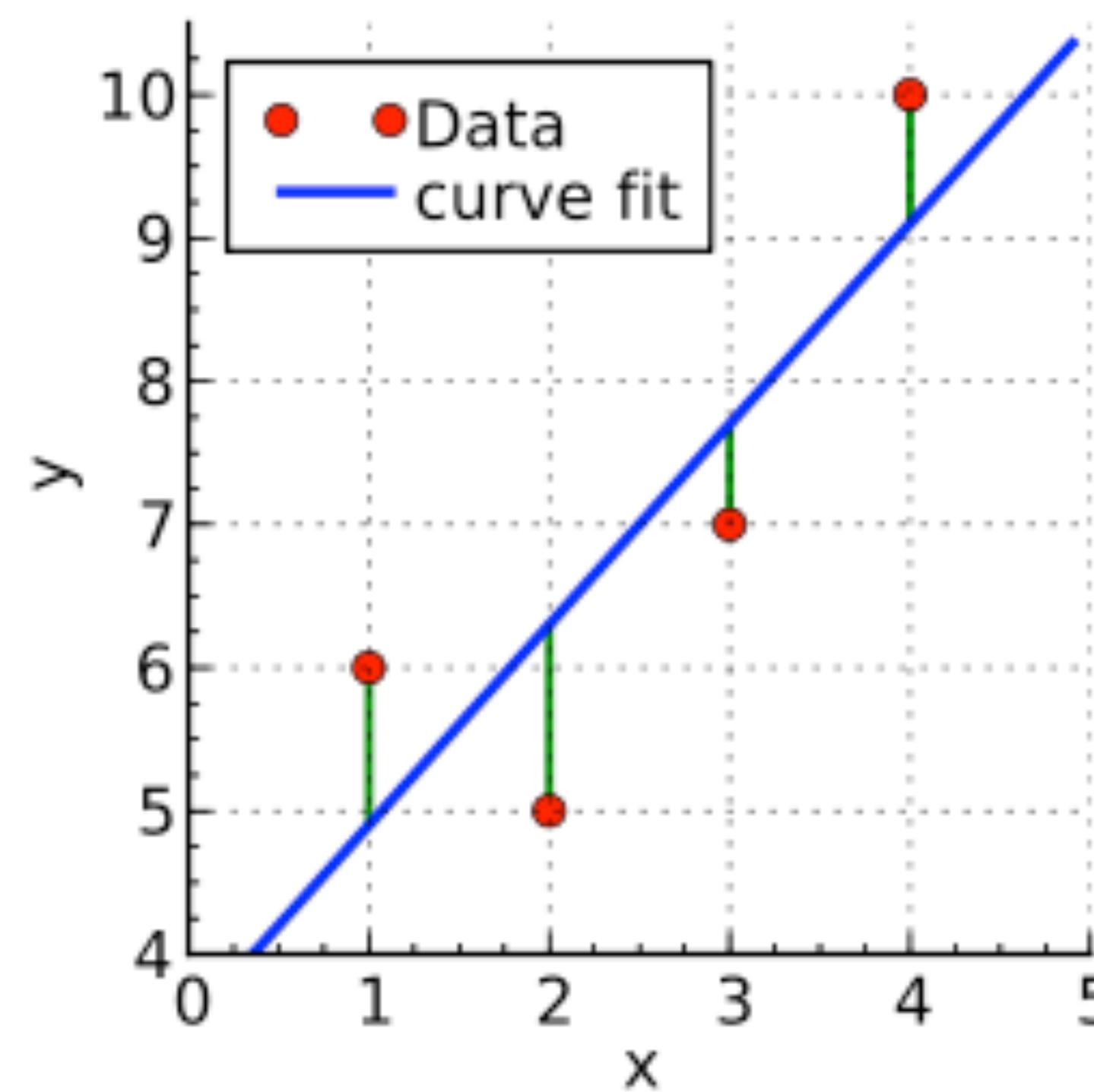
# visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()
```

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

Linear Regression (Line/Plane Fitting)



Linear Regression (Line/Plane Fitting)



LS Solution for Regression

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

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

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

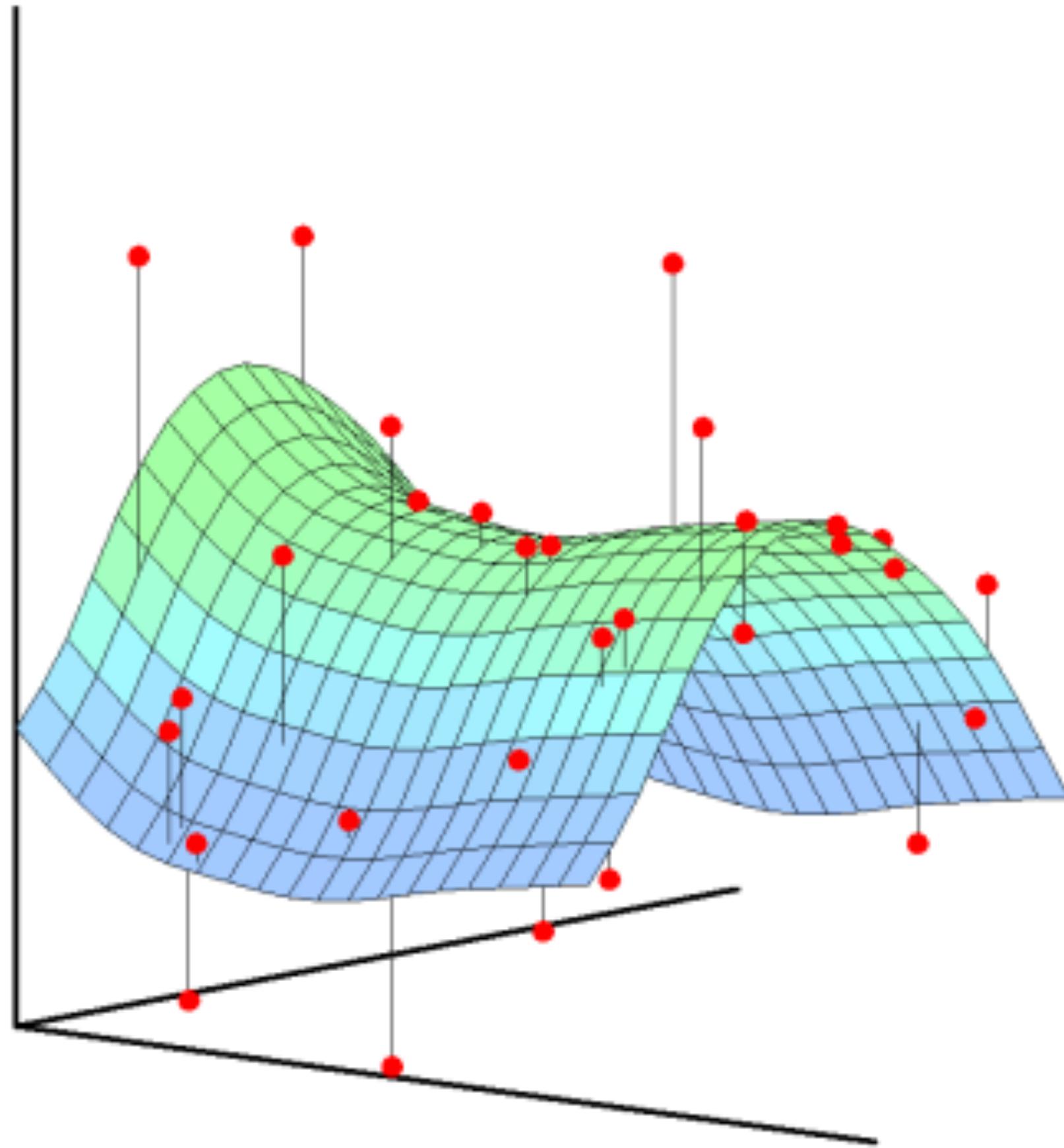
LS Solution for Regression

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

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

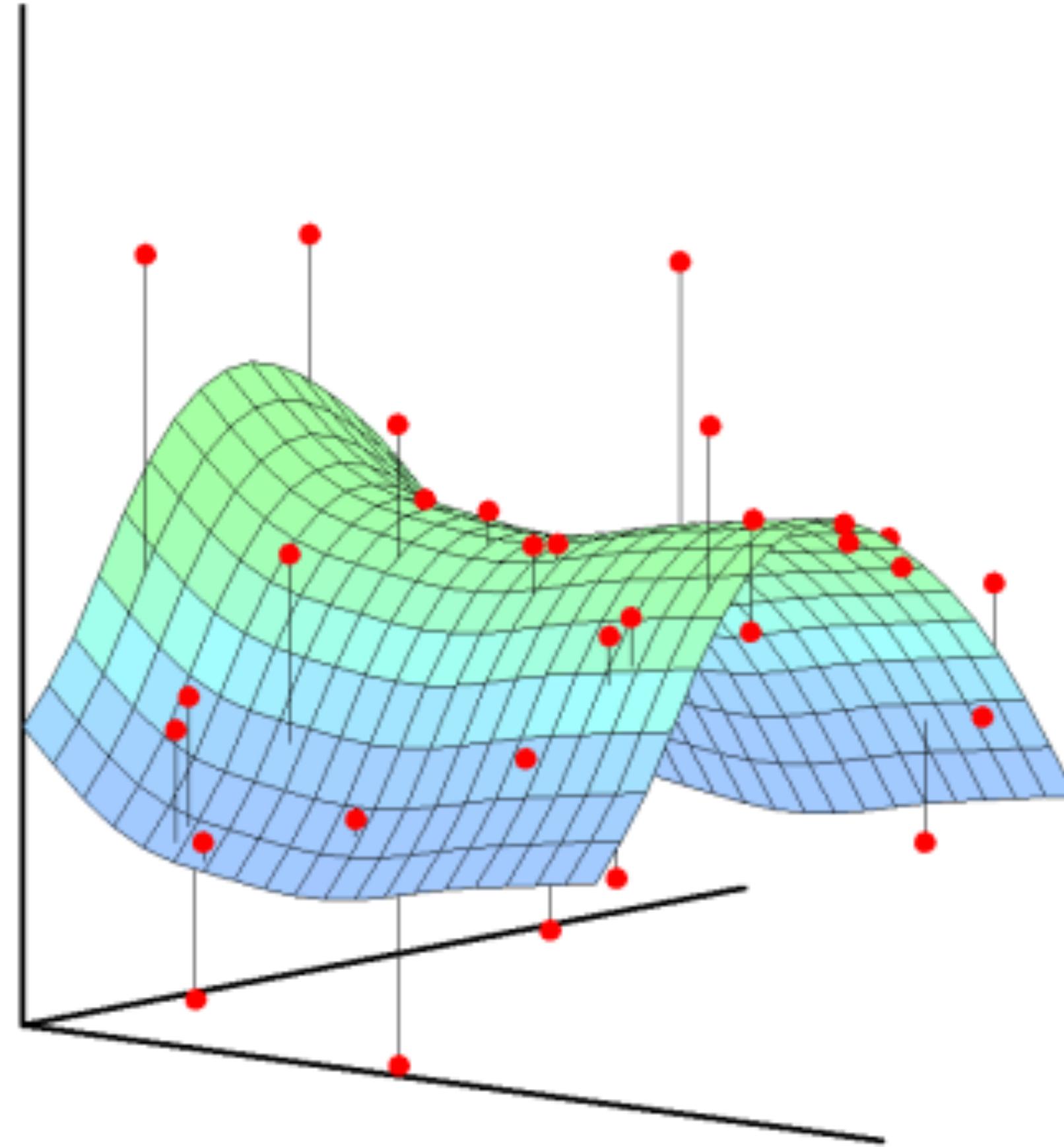
$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \quad \mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$$

Generalized Linear Regression



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

Generalized Linear Regression

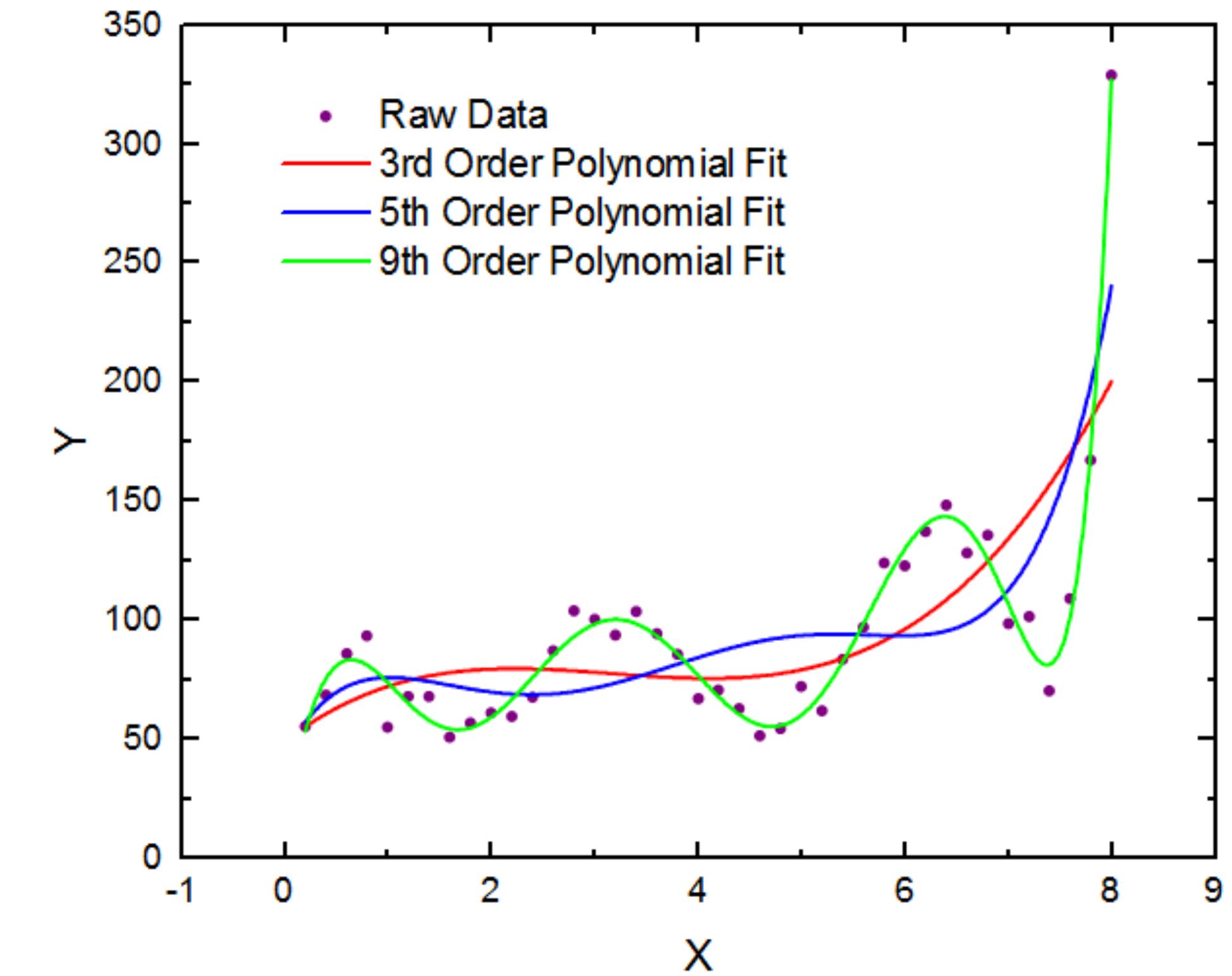


$$\mathbf{x} \rightarrow \phi(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}$$

known nonlinearity

1D Example: k-th Degree Polynomial Fitting

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



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

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

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

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

Nx1 **NxM** **Mx1** **Nx1**

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

LS Solution for Linear Regression

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

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

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

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

LS Solution for Generalized Linear Regression

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

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

LS Solution for Generalized Linear Regression

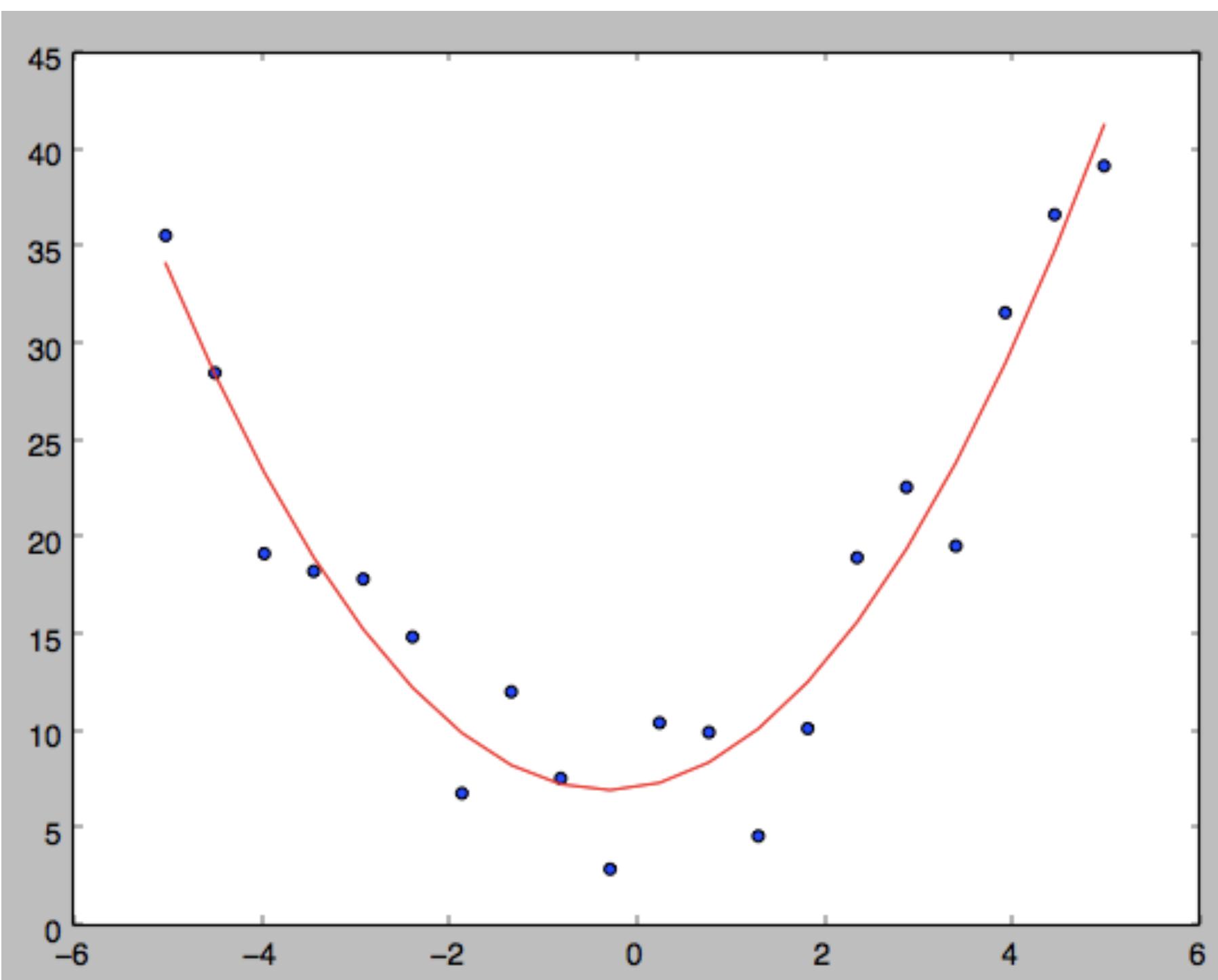
$$\mathbf{y} = \Phi \mathbf{w} + \epsilon$$

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

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

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

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))

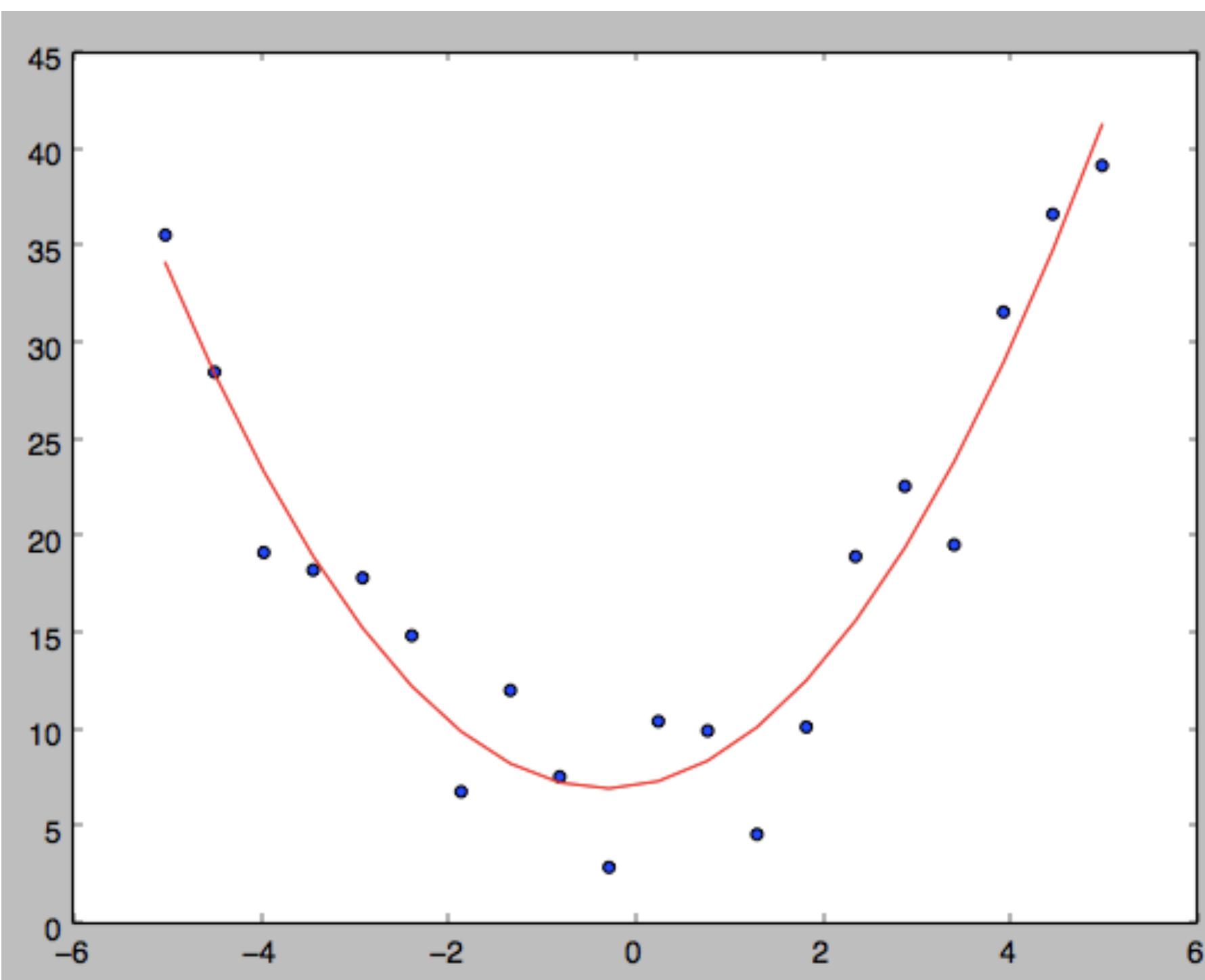
# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1), (x**2).reshape(-1, 1)] )

# 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)] )

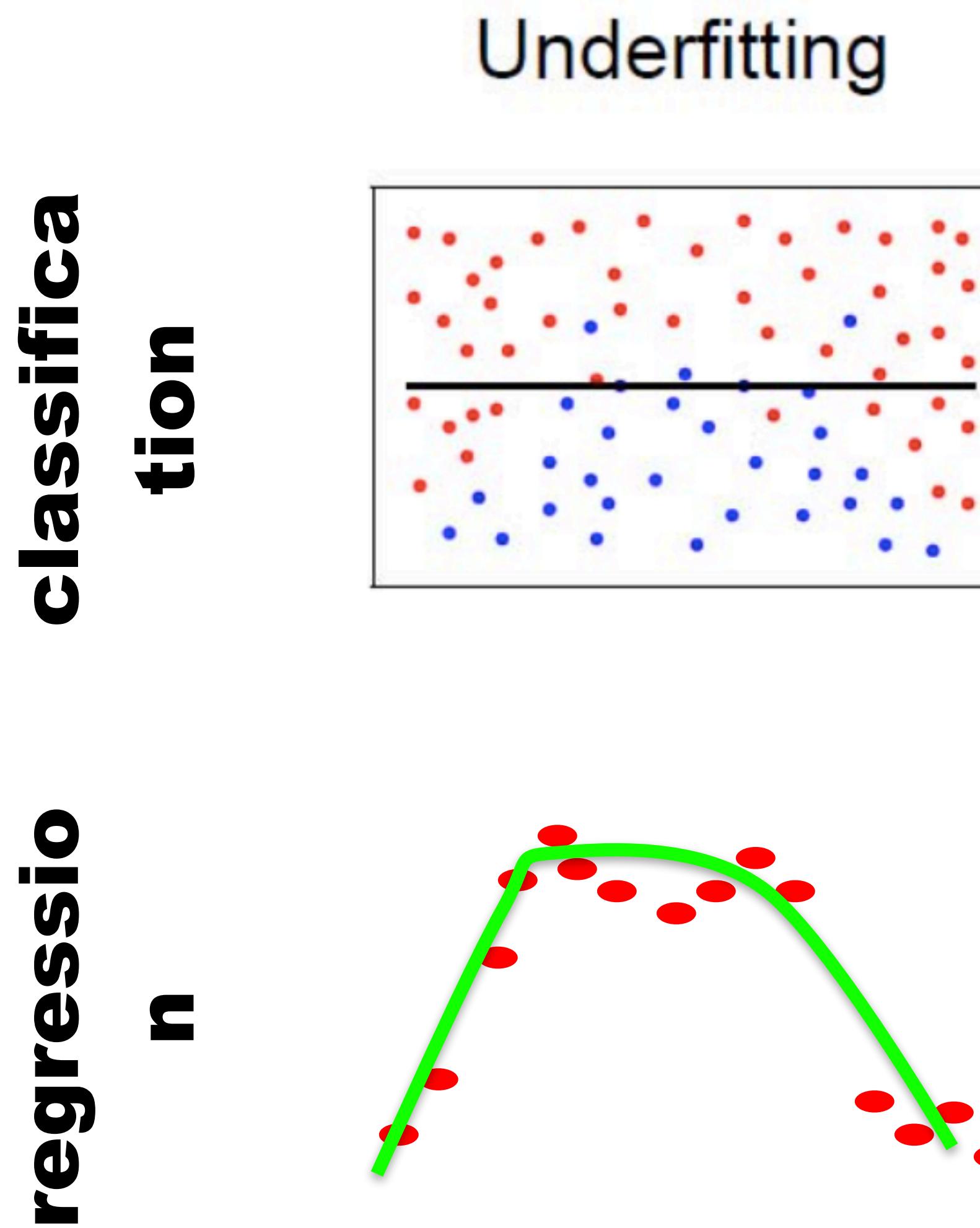
# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# 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

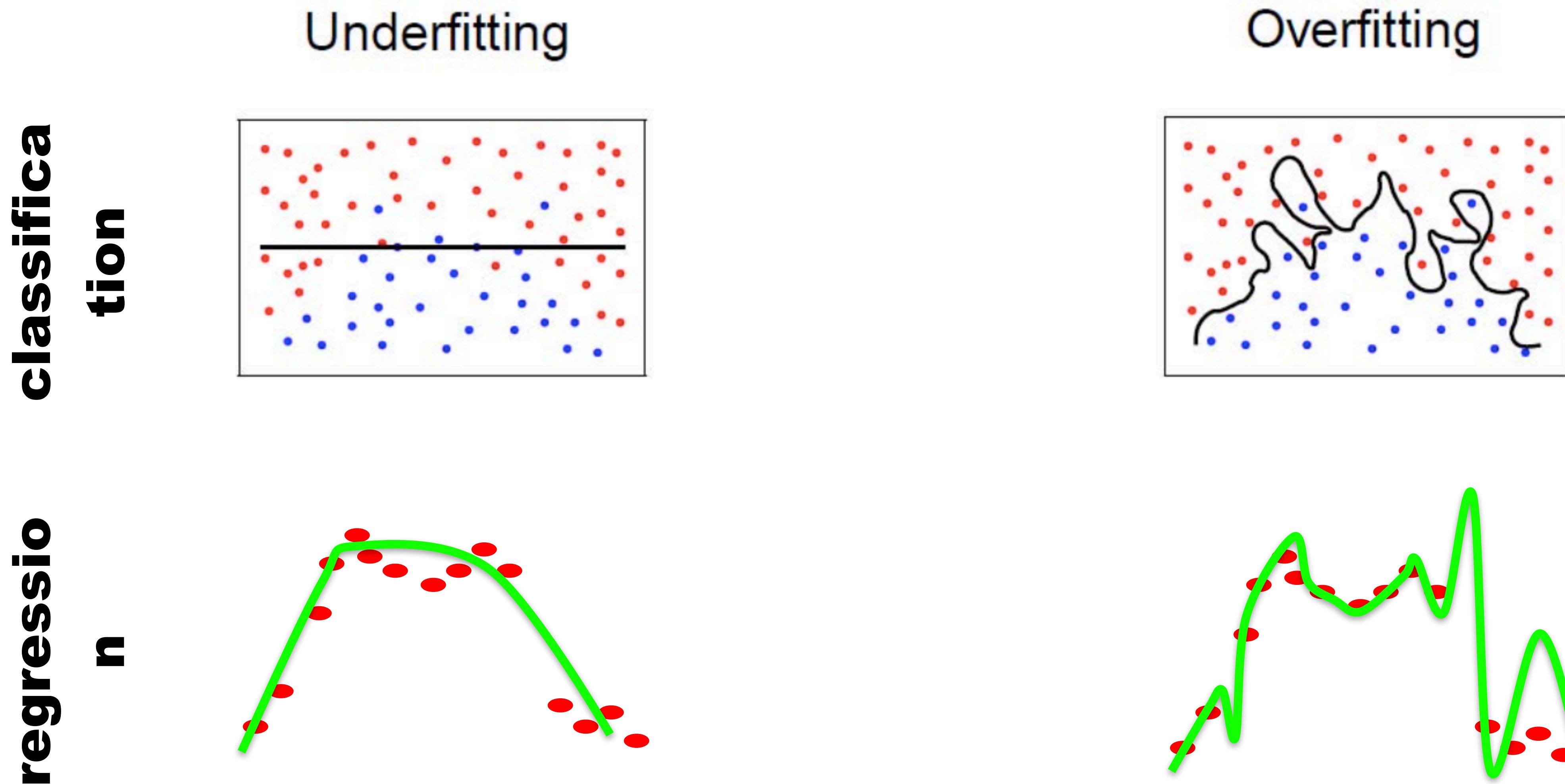
# visualize the fitted model
pyplot.scatter(x, y)
pyplot.plot(x, y_est, color='red')
pyplot.show()
```

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

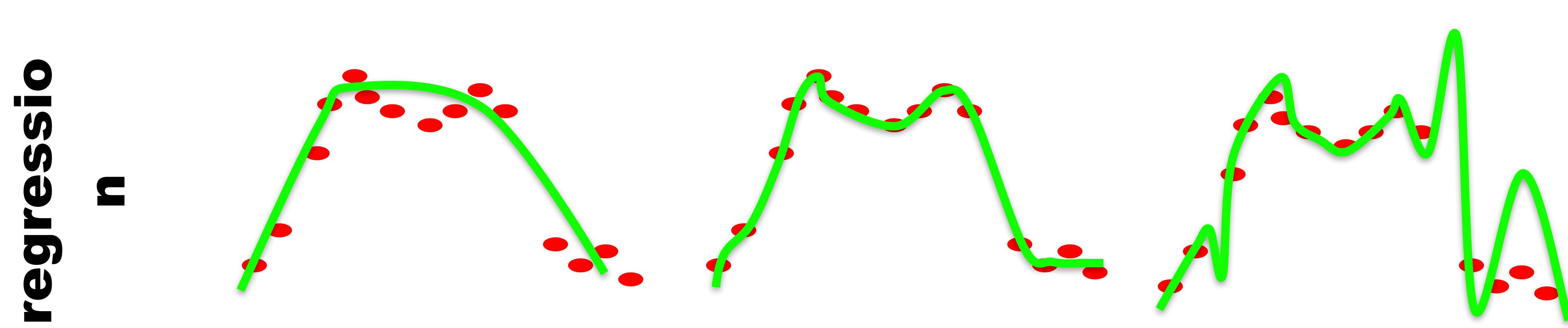
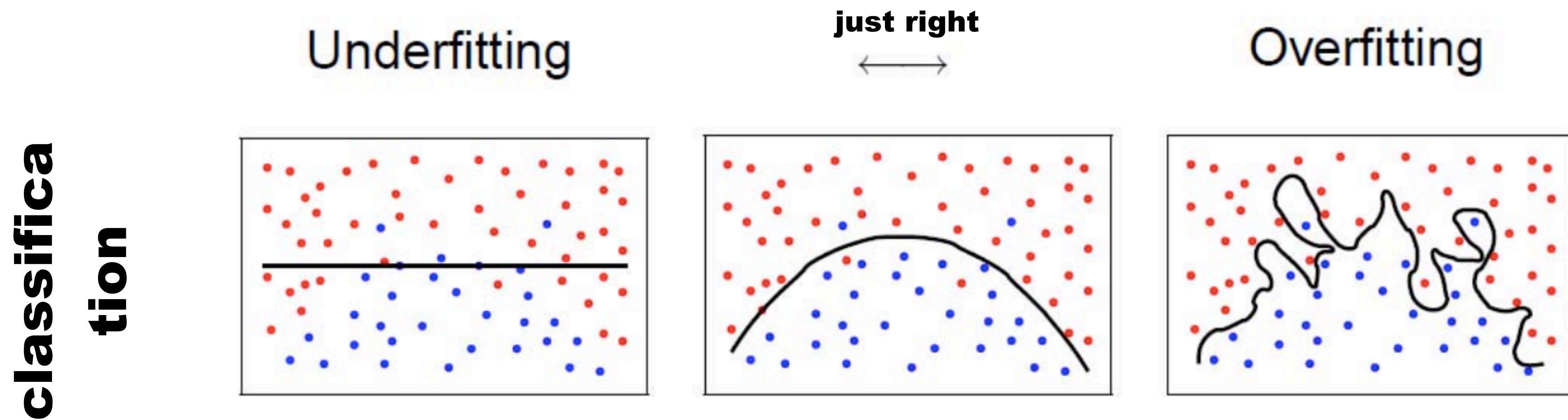
Underfitting vs. Overfitting



Underfitting vs. Overfitting



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

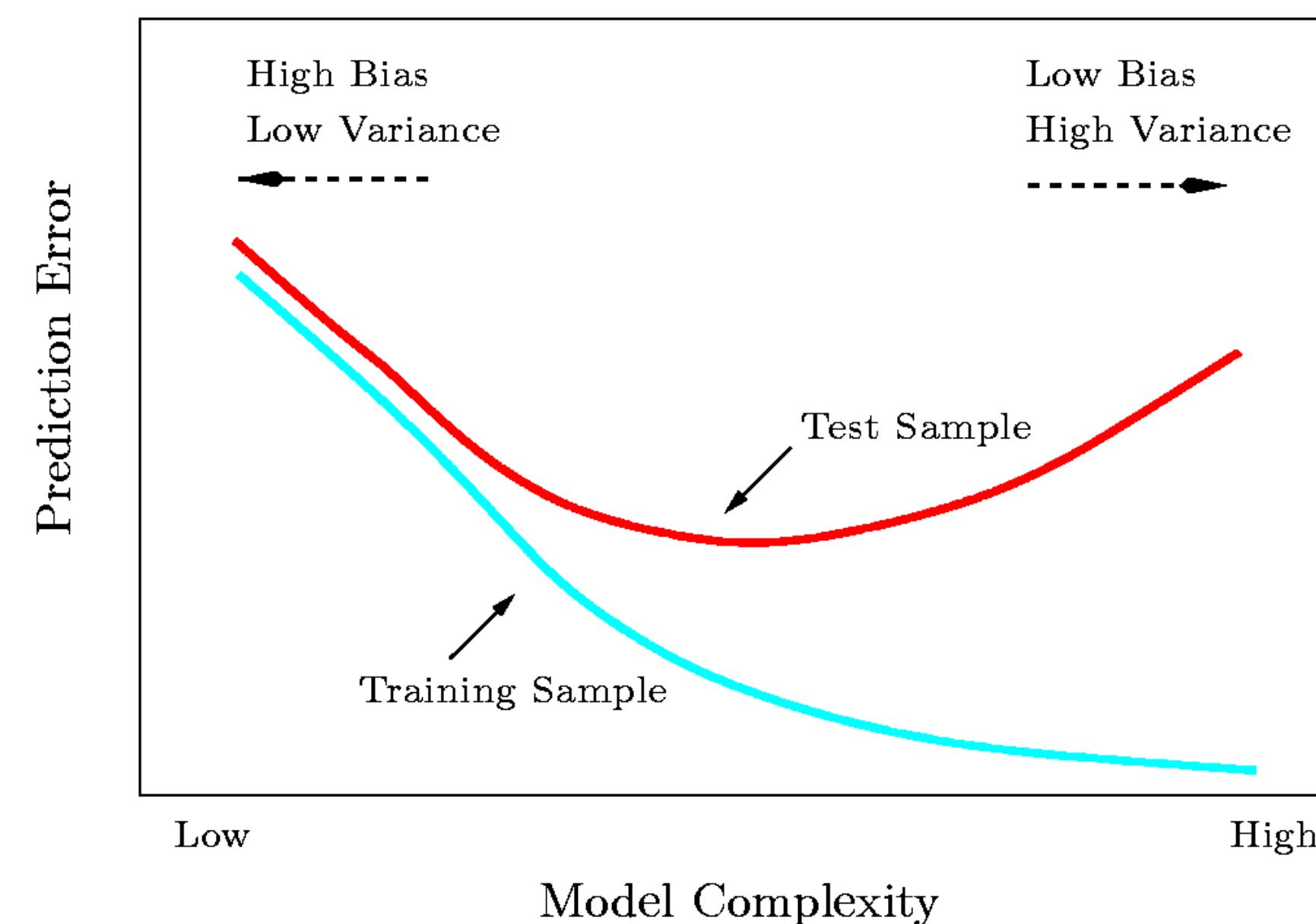
A *flexible model* approximates the target function well in the training set but can “overtrain” and have poor performance on the test set (“variance”).

A *rigid model* 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* is more predictable in the test set but the model may not be good even on the training set (“bias”).



Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

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

Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

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

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

Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

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

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

“data fidelity”

Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

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

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

“data fidelity” **complexity**

Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

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

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

↑ **“data fidelity”** **complexity**

minimum remains to be determined

Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

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

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



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

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

Least Squares Solution

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

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

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

Condition for minimum:

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

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

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:

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

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

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

Ridge regression: L2-regularized Linear Regression

$$\begin{aligned} L(\mathbf{w}) &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + \lambda \mathbf{w}^T \mathbf{w} \\ &= \mathbf{y}^T \mathbf{y} - 2 \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w}^T \mathbf{I} \mathbf{w} \\ &\quad \text{as before, for linear regression} \qquad \text{identity matrix} \\ &= \mathbf{y}^T \mathbf{y} - 2 \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{w}^T (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} \end{aligned}$$

Condition for minimum:

$$\begin{aligned}\nabla L(\mathbf{w}^*) &= 0 \\ -2\mathbf{X}^T \mathbf{y} + 2(\mathbf{X}^T \mathbf{X} + \lambda I) \mathbf{w}^* &= 0 \\ \mathbf{w}^* &= (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T \mathbf{y}\end{aligned}$$

Ridge regression: L2-regularized Linear Regression

$$\begin{aligned} L(\mathbf{w}) &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + \lambda \mathbf{w}^T \mathbf{w} \\ &= \mathbf{y}^T \mathbf{y} - 2 \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w}^T \mathbf{I} \mathbf{w} \\ &\quad \text{as before, for linear regression} \qquad \text{identity matrix} \\ &= \mathbf{y}^T \mathbf{y} - 2 \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{w}^T (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} \end{aligned}$$

Condition for minimum:

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

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

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

Ridge regression: L2-regularized Linear Regression

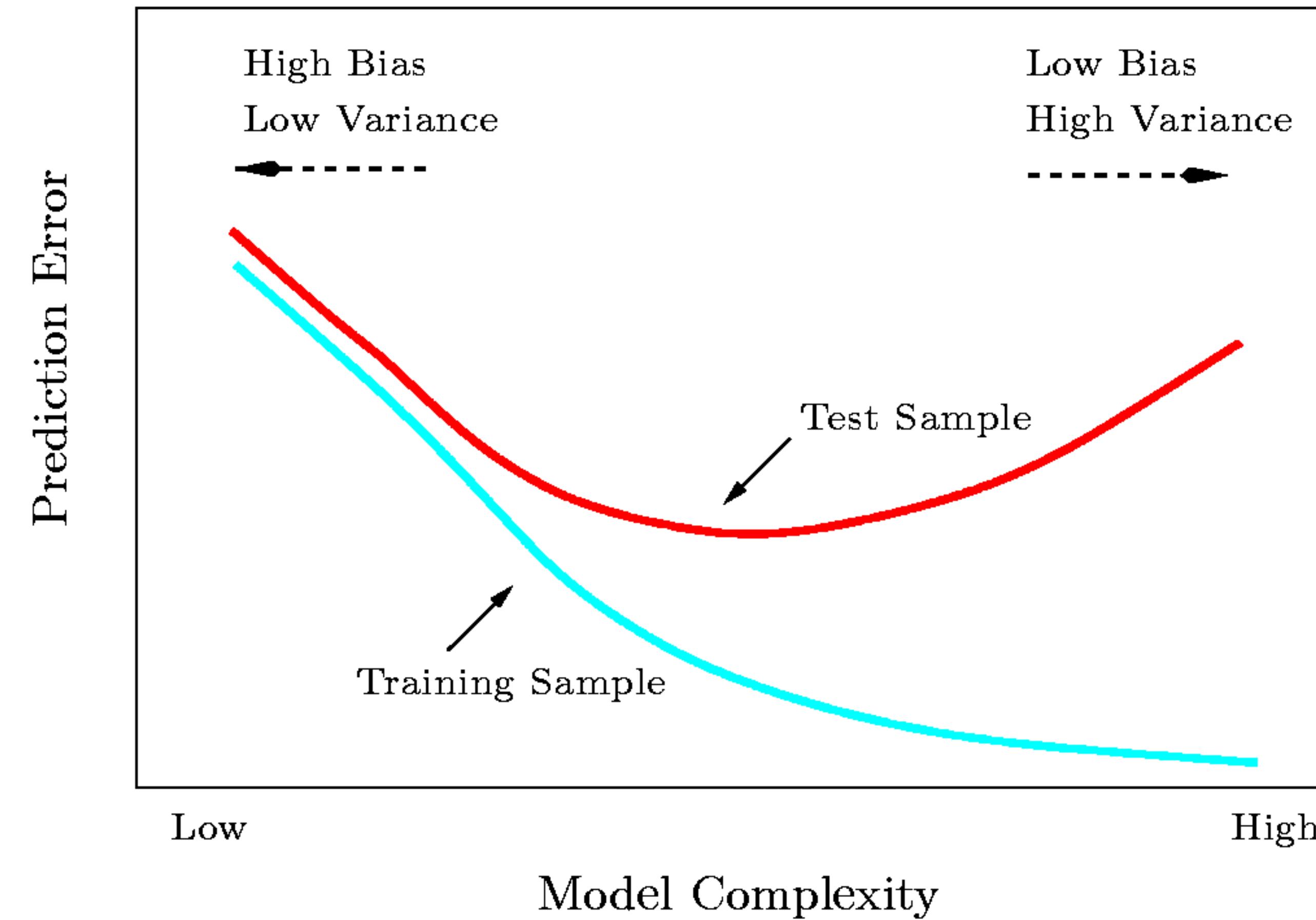
$$\begin{aligned} L(\mathbf{w}) &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + \lambda \mathbf{w}^T \mathbf{w} \\ &= \mathbf{y}^T \mathbf{y} - 2 \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w}^T \mathbf{I} \mathbf{w} \\ &\quad \text{as before, for linear regression} \qquad \text{identity matrix} \\ &= \mathbf{y}^T \mathbf{y} - 2 \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{w}^T (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} \end{aligned}$$

Condition for minimum:

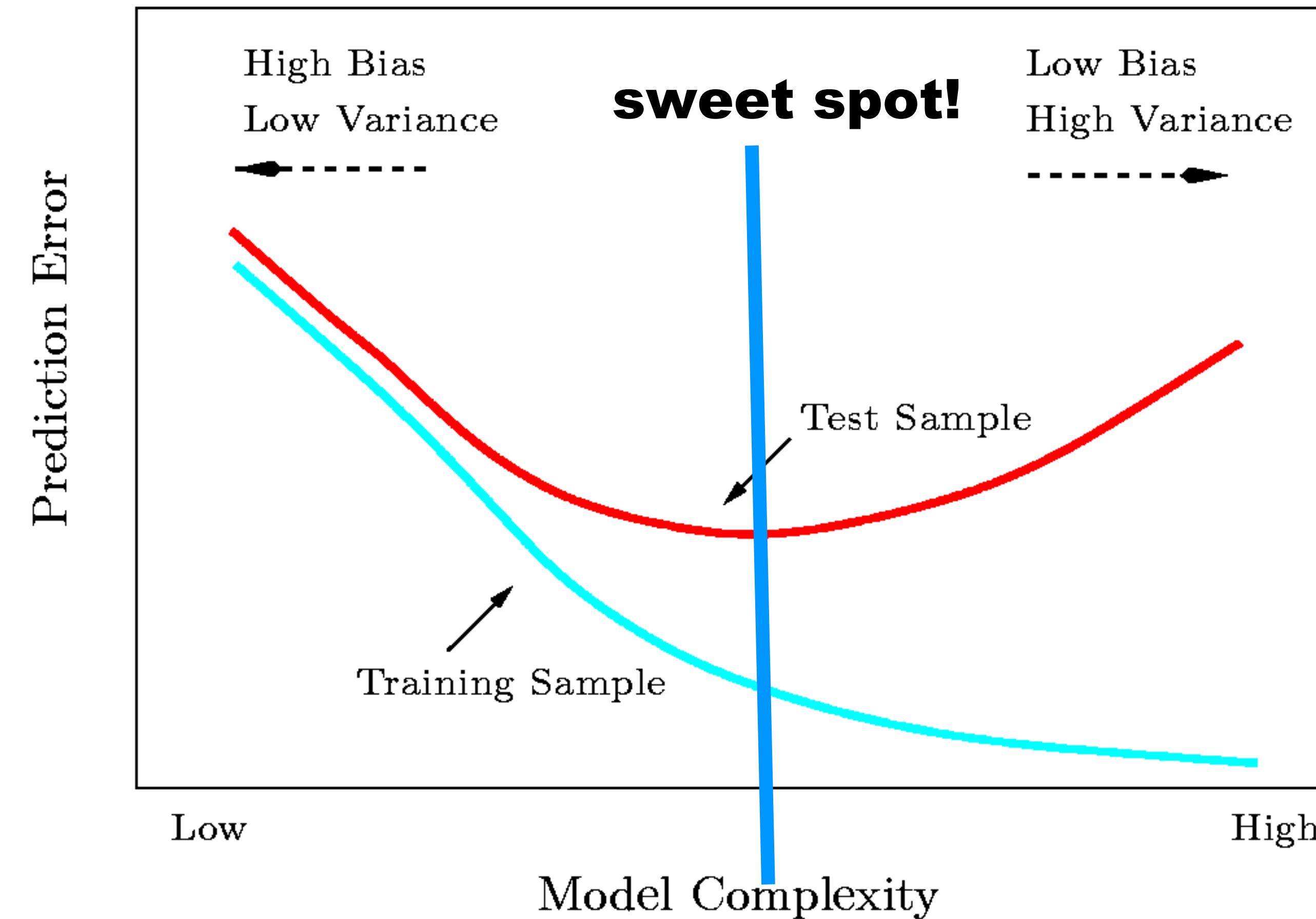
$$\nabla L(\mathbf{w}^*) = 0$$
$$-2\mathbf{X}^T \mathbf{y} + 2(\mathbf{X}^T \mathbf{X} + \lambda I) \mathbf{w}^* = 0$$
$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \boxed{\lambda I})^{-1} \mathbf{X}^T \mathbf{y}$$

λ : "hyperparameter"
 $y = 0$

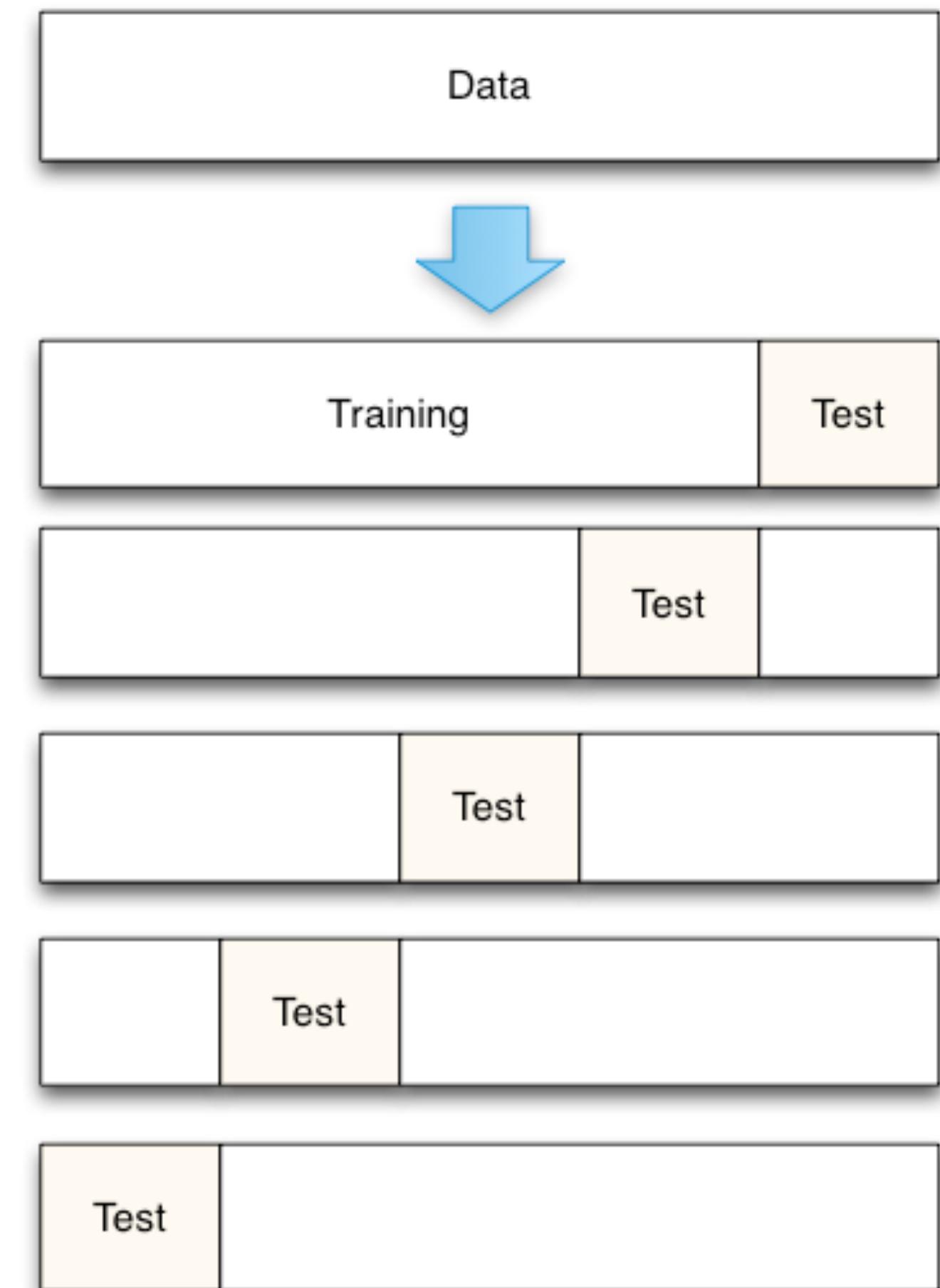
Bias-Variance Tradeoff (function of λ)



Bias-Variance Tradeoff (function of λ)



Selecting λ with Cross-validation

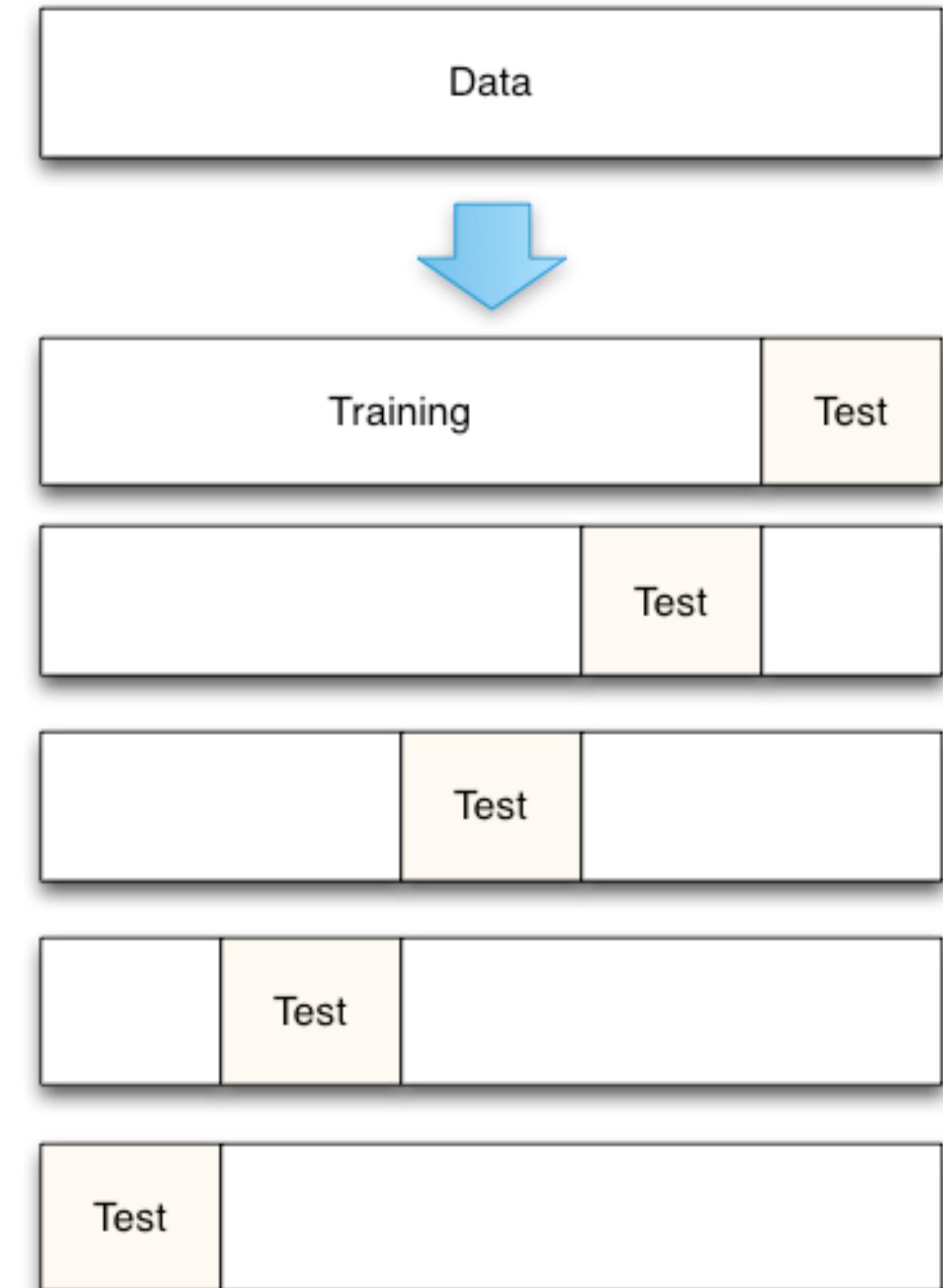


Selecting λ with Cross-validation

- **Cross validation technique**

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

- **K-fold cross validation:**



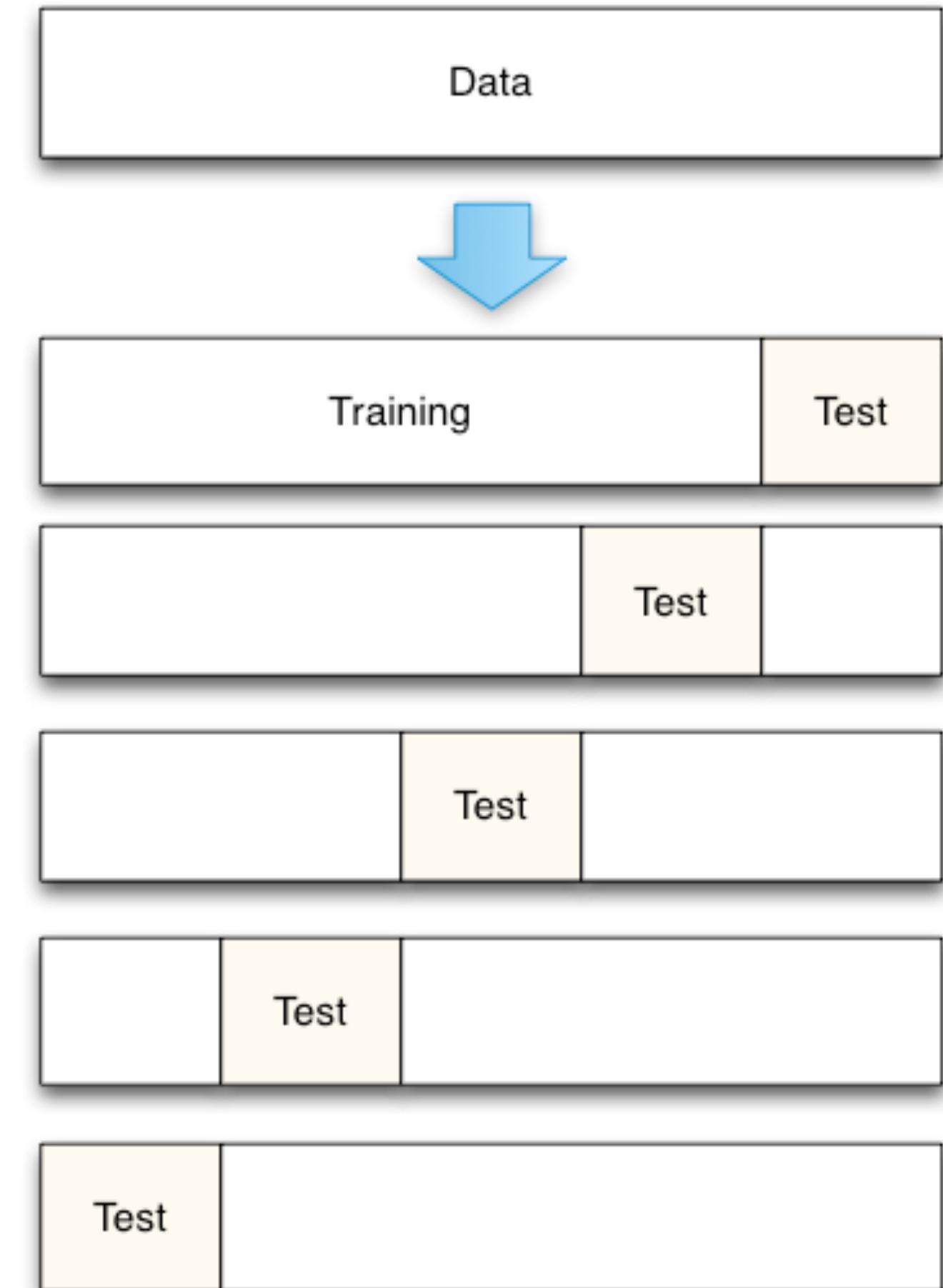
Selecting λ with Cross-validation

- **Cross validation technique**

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

- **K-fold cross validation:**

- K splits, average K errors



Selecting λ with Cross-validation

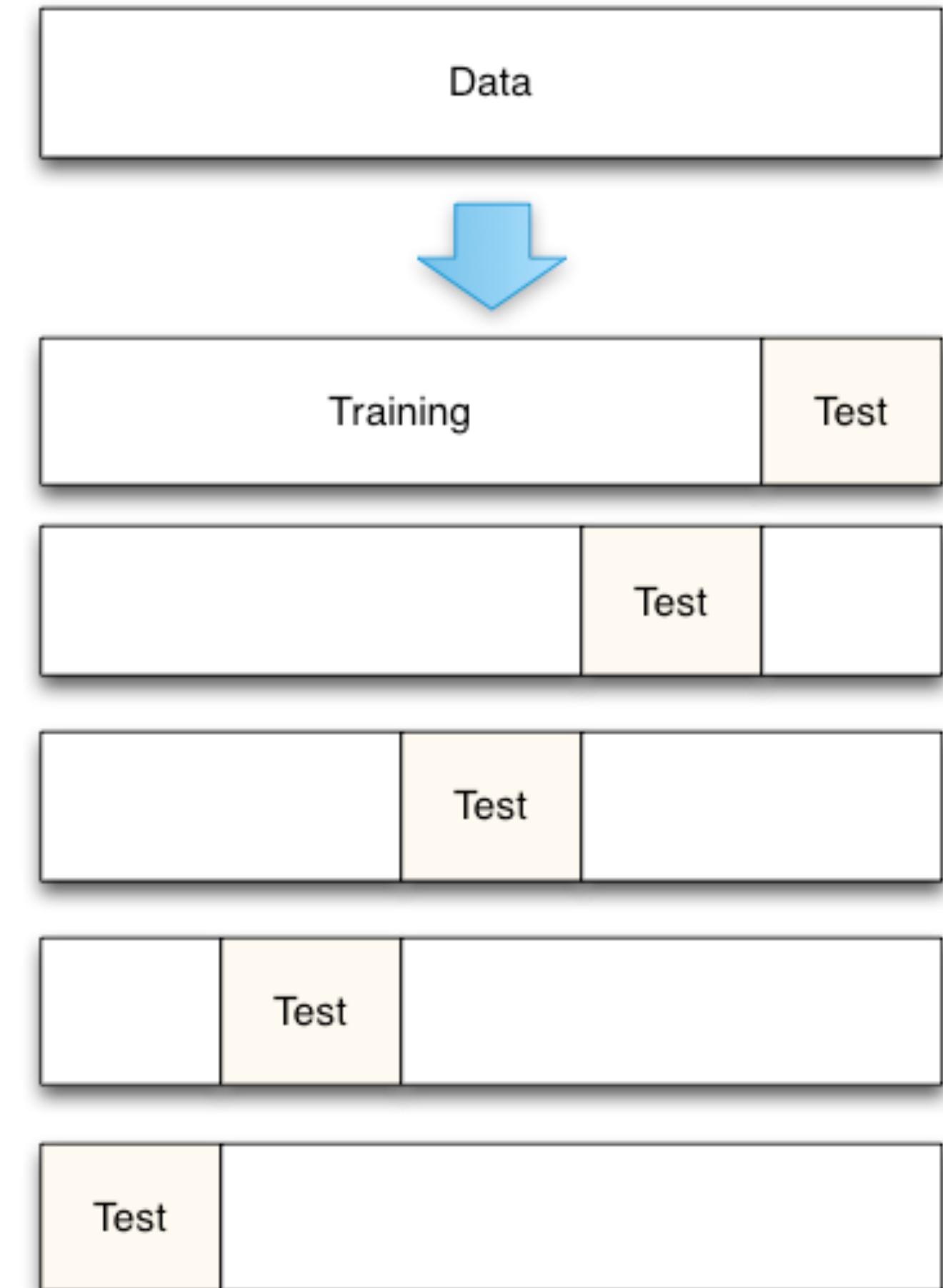
- **Cross validation technique**

- 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 λ**



Selecting λ with Cross-validation

- **Cross validation technique**

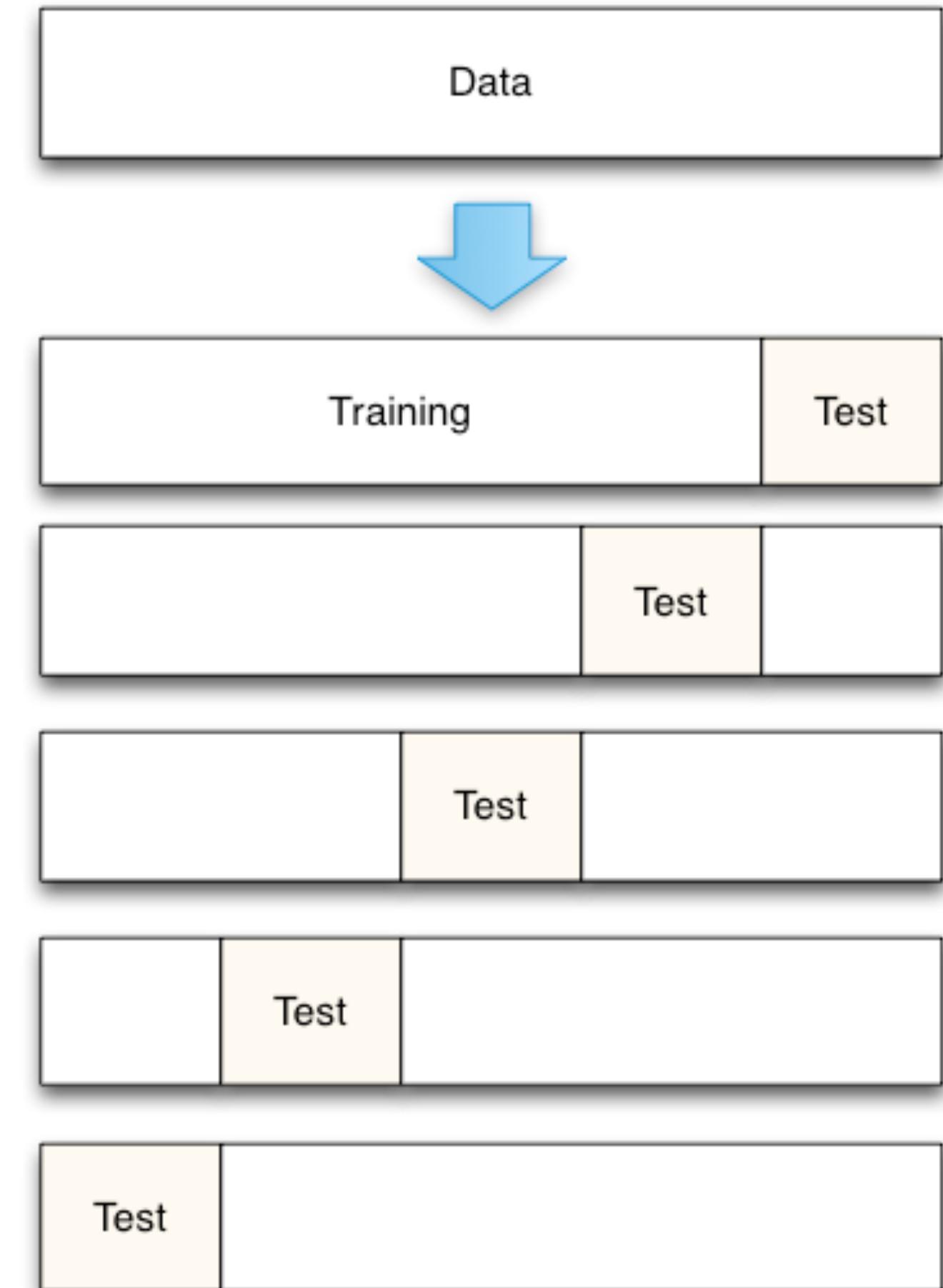
- Exclude part of the training data from parameter estimation
- Use them only to predict the test error

- **K-fold cross validation:**

- K splits, average K errors

- **Use cross-validation for different 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

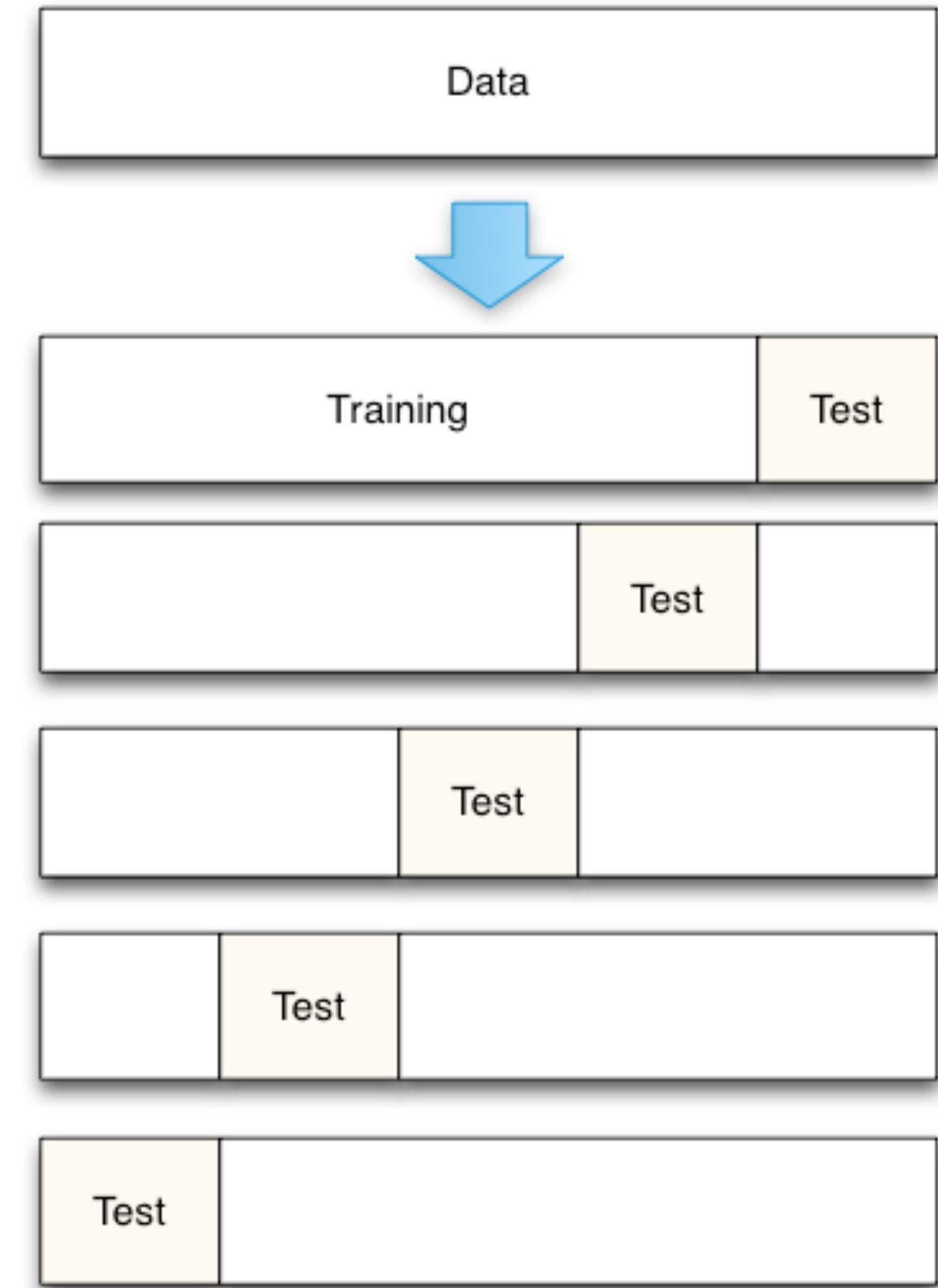
- **K-fold cross validation:**

- K splits, average K errors

- **Use cross-validation for different λ**

- pick value that minimizes cross-validation error

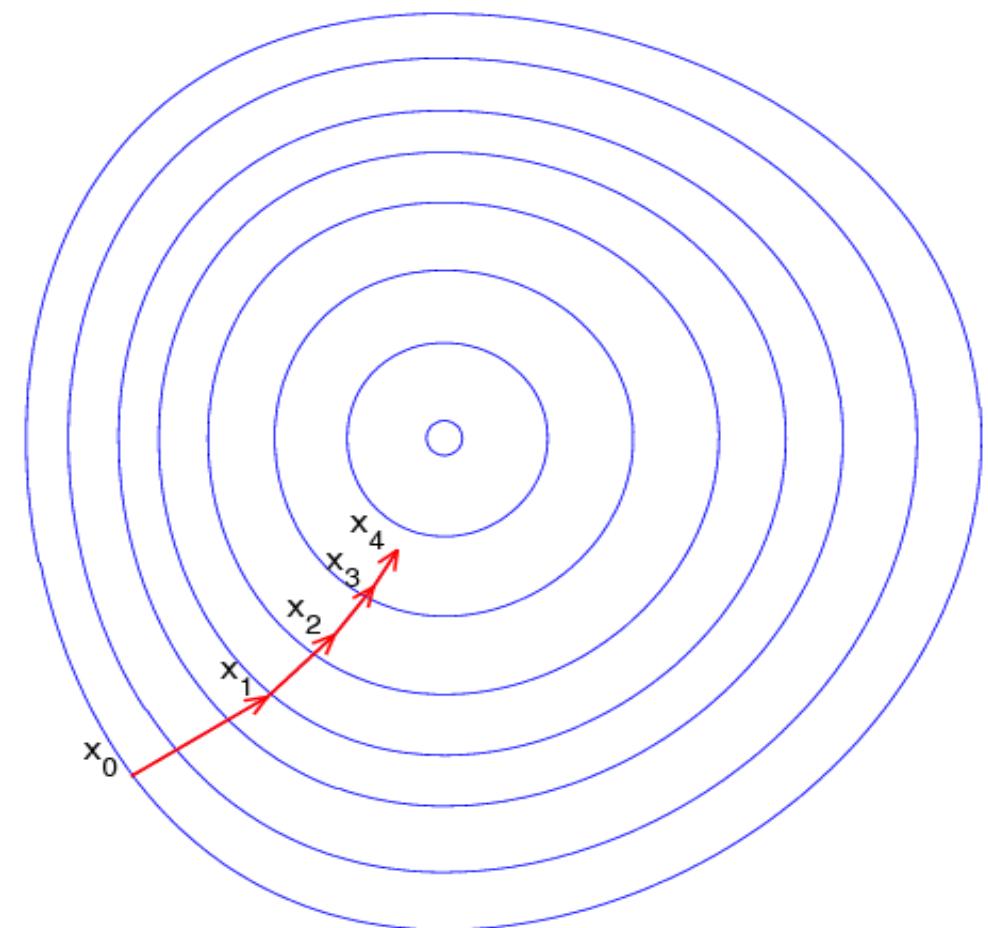
Least glorious, most effective of all methods



Gradient Descent Minimization

Initialize: \mathbf{x}_0

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

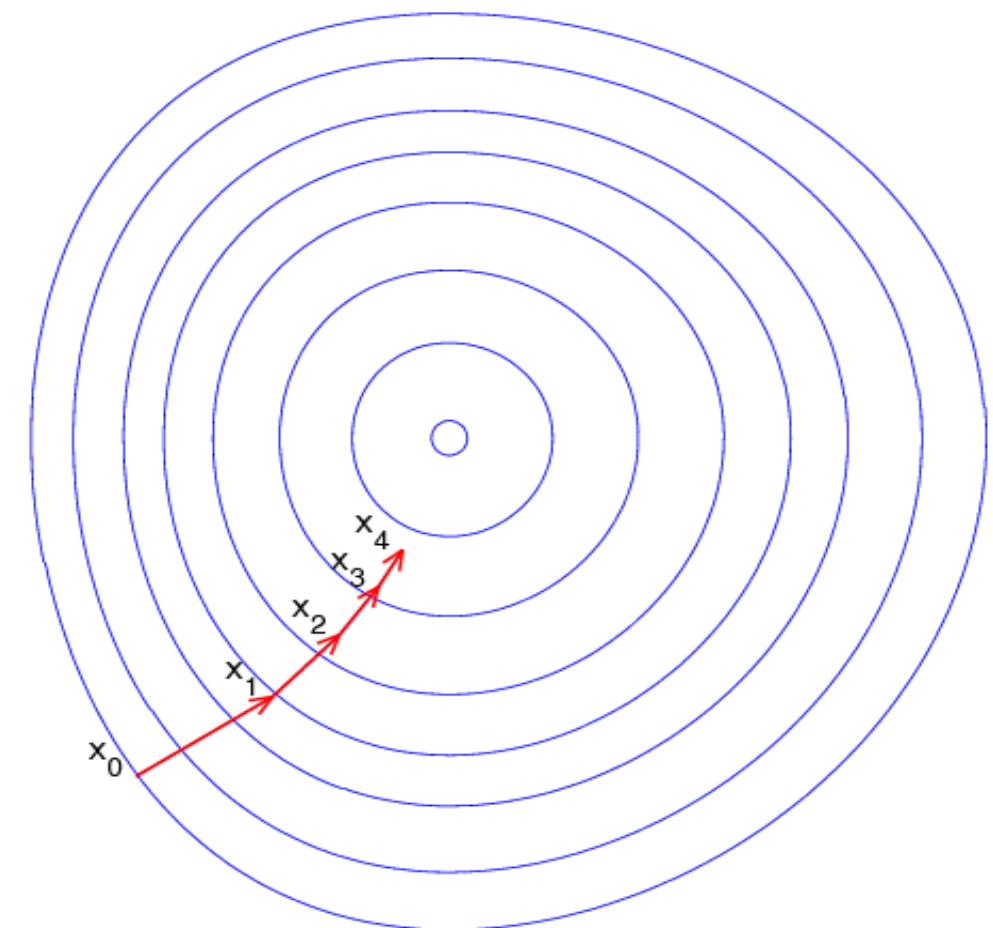


Gradient Descent Minimization

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.

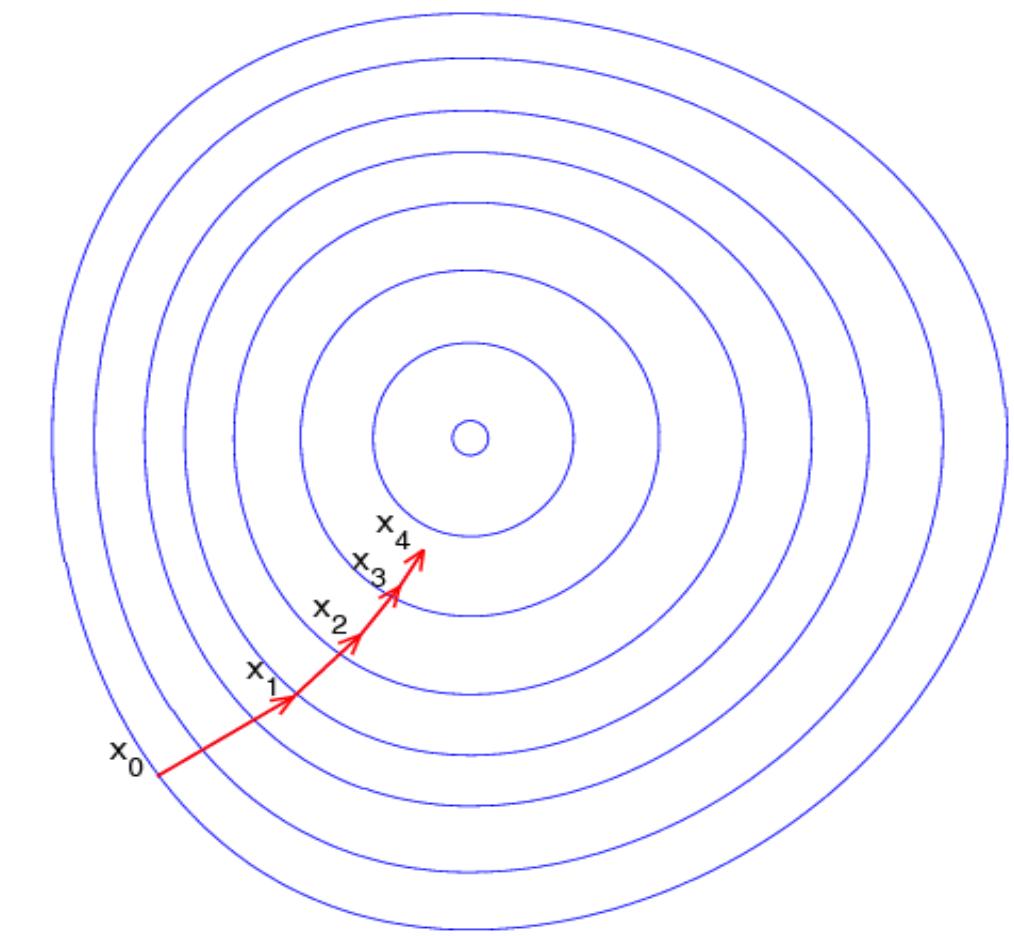
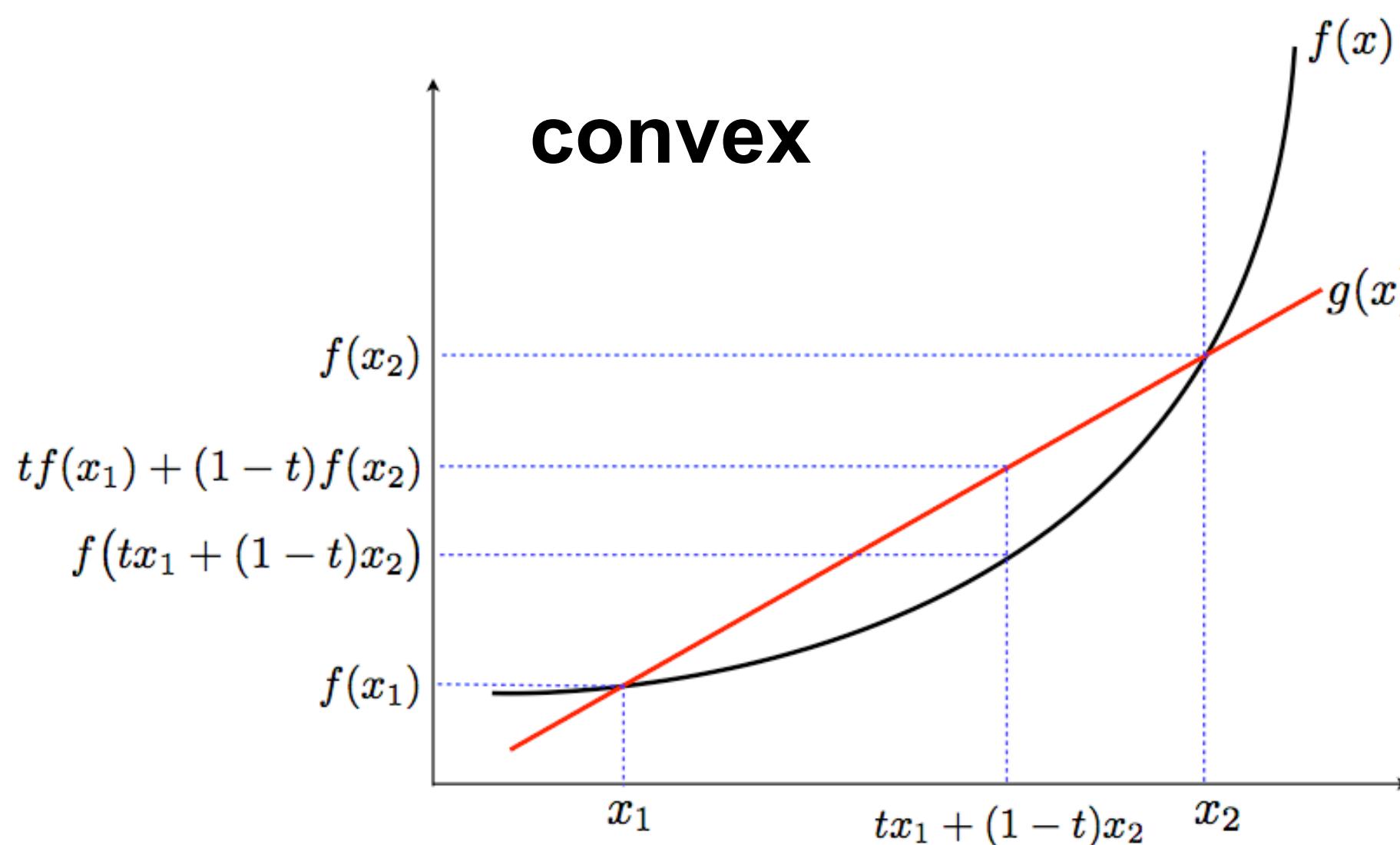


Gradient Descent Minimization

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.

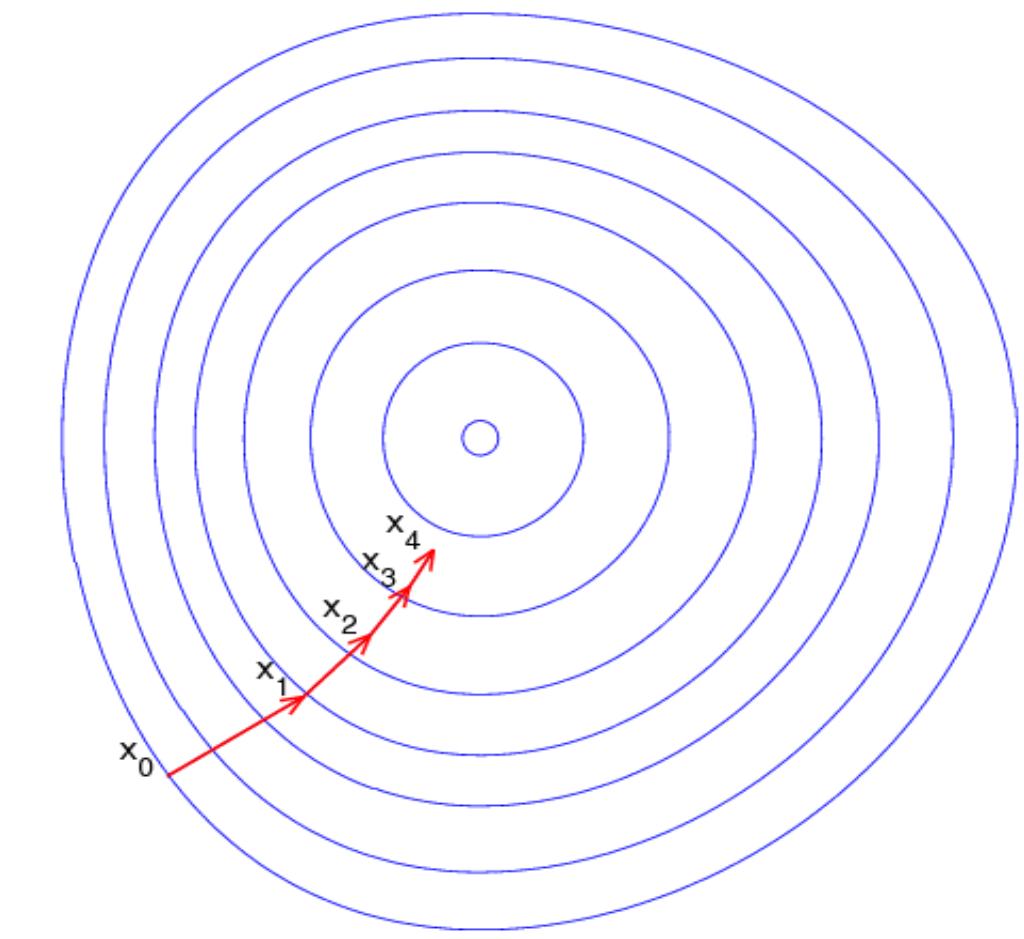
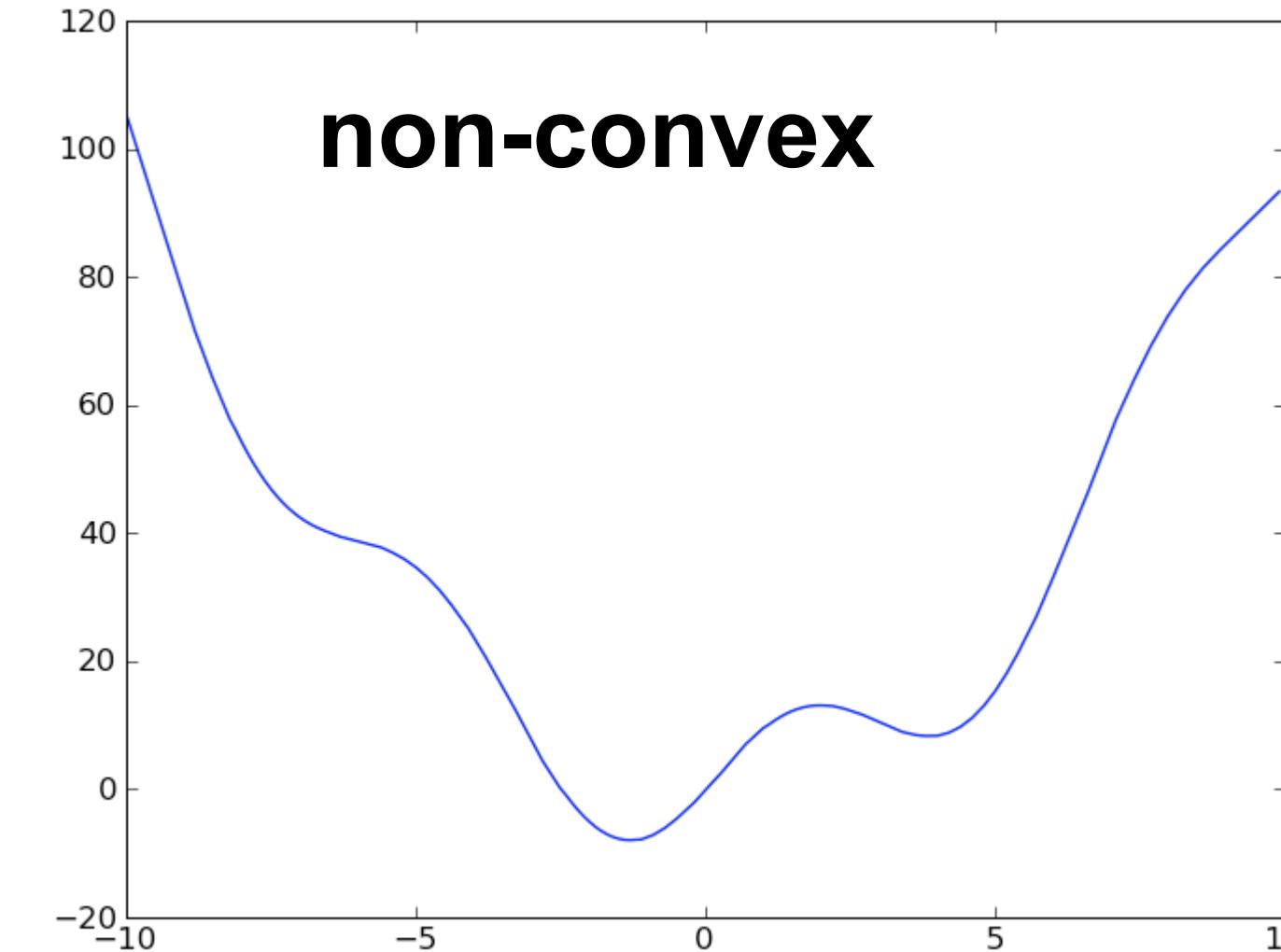
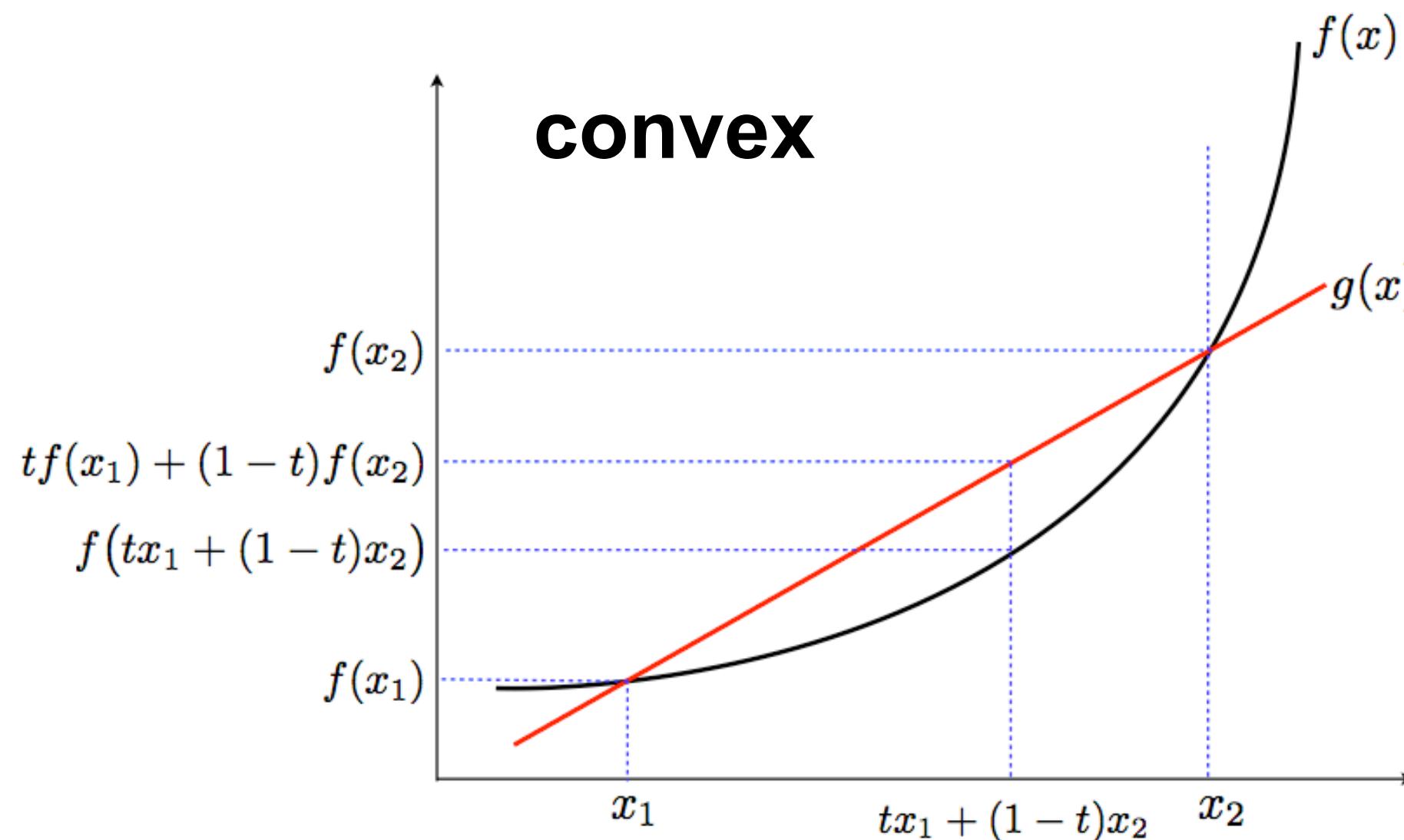


Image Classifier

Image Classifier

- **Data (training/validation/test)**

$$\{\mathbf{x}_i, y_i\}$$

Image Classifier

- Data (training/validation/test)

$$\{\mathbf{x}_i, y_i\}$$

- Scoring function
(probability of belonging to class k)

$$f(\mathbf{x}_i, \mathbf{W})$$

Image Classifier

- **Data (training/validation/test)**

$$\{\mathbf{x}_i, y_i\}$$

- **Scoring function
(probability of belonging to class k)**

$$f(\mathbf{x}_i, \mathbf{W})$$

- **Loss function** $L_i(W) := h(f(\mathbf{x}_i, W), y_i)$ $L(W) := \frac{1}{N} \sum_i L_i(W)$

Image Classifier

- **Data (training/validation/test)**

$$\{\mathbf{x}_i, y_i\}$$

- **Scoring function
(probability of belonging to class k)**

$$f(\mathbf{x}_i, \mathbf{W})$$

- **Loss function** $L_i(W) := h(f(\mathbf{x}_i, W), y_i)$ $L(W) := \frac{1}{N} \sum_i L_i(W)$

- **Optimization** $\mathbf{W}^* := \arg \min L(W) = \arg \min L_i(W)/N$

Linear versus Nonlinear Models

$$s = f(\mathbf{x}_i, W) = W\mathbf{x}_i$$

Linear versus Nonlinear Models

- **Objective function #1** $s = f(\mathbf{x}_i, W) = W\mathbf{x}_i$
- $$L_i(W) := (Wx_i - y_i)^2$$

Linear versus Nonlinear Models

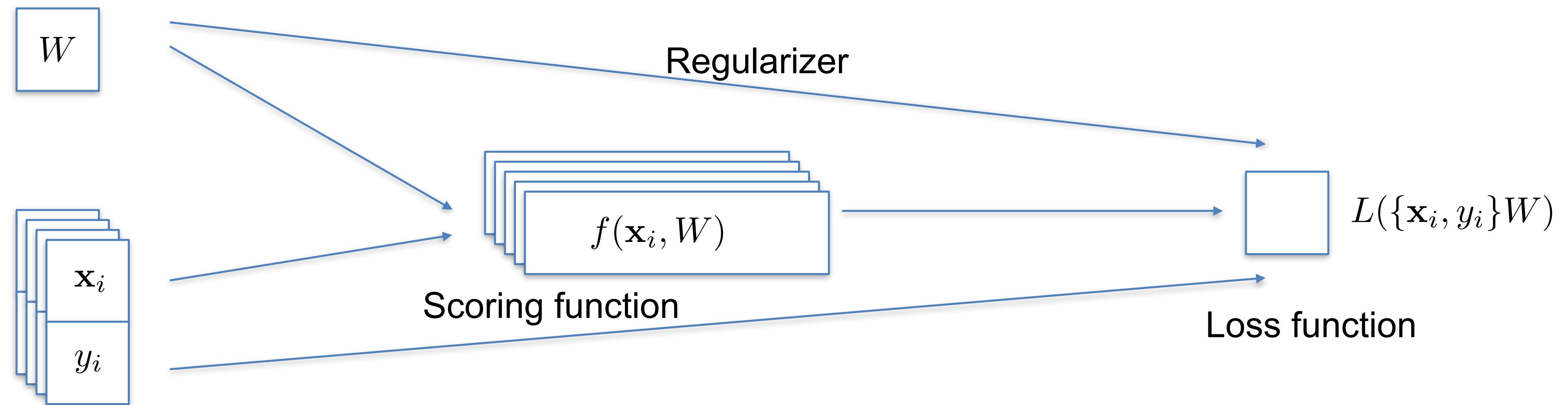
- **Objective function #1** $s = f(\mathbf{x}_i, W) = W\mathbf{x}_i$

$$L_i(W) := (Wx_i - y_i)^2$$

- **Objective function #2**

$$L_i(W) := -\log \left(\frac{e^{Wx_i y_i}}{\sum_j e^{Wx_j y_j}} \right) = -\log \left(\frac{e^{sy_i}}{\sum_j e^{sy_j}} \right)$$

Image Classifier



Method 1a: Random search

- `bestLoss = infinity`

- `Loop`

 - `Pick random W`

 - `Compute loss`

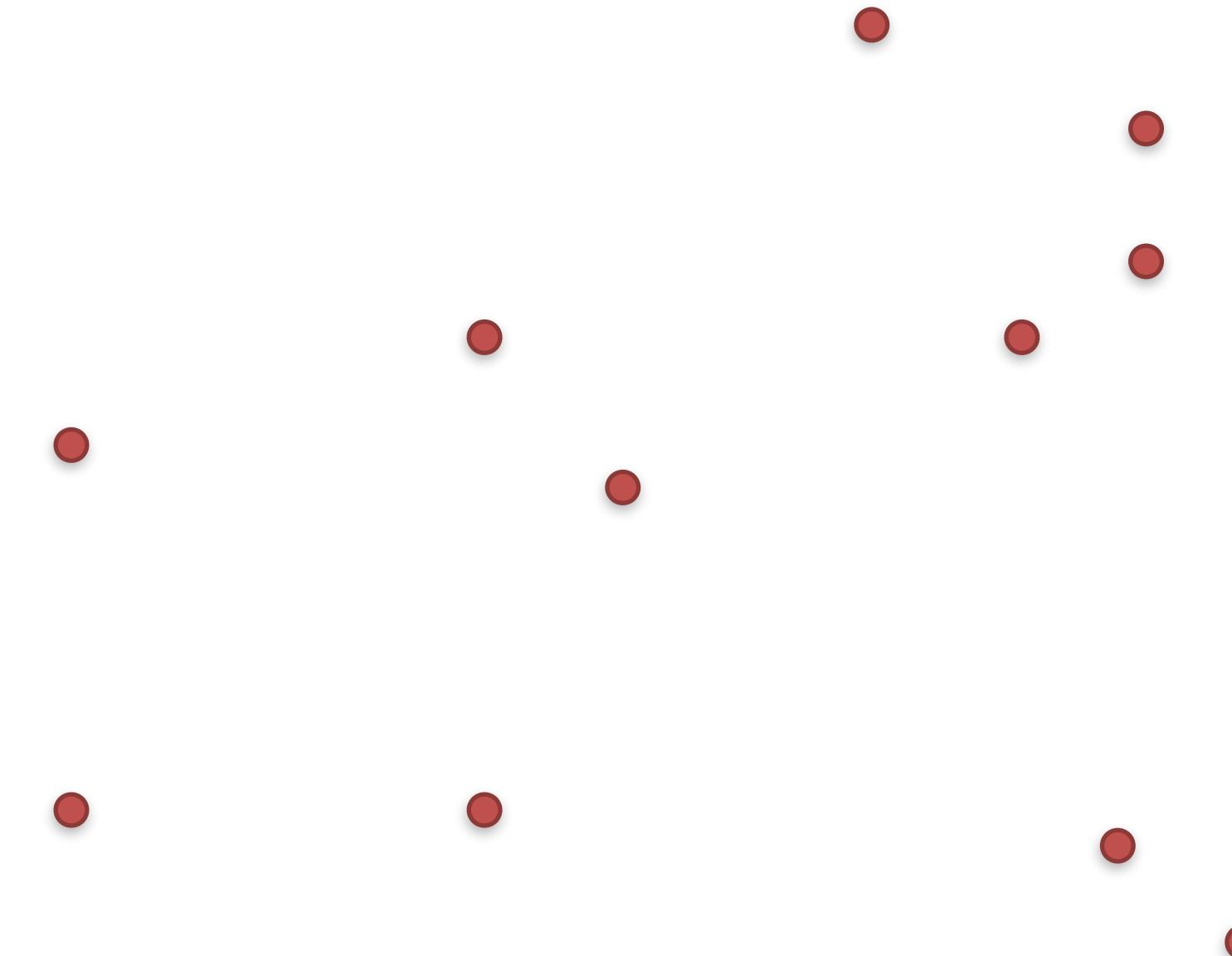
 - `if loss < bestLoss`

 - `bestLoss = loss`

 - `currW = W`

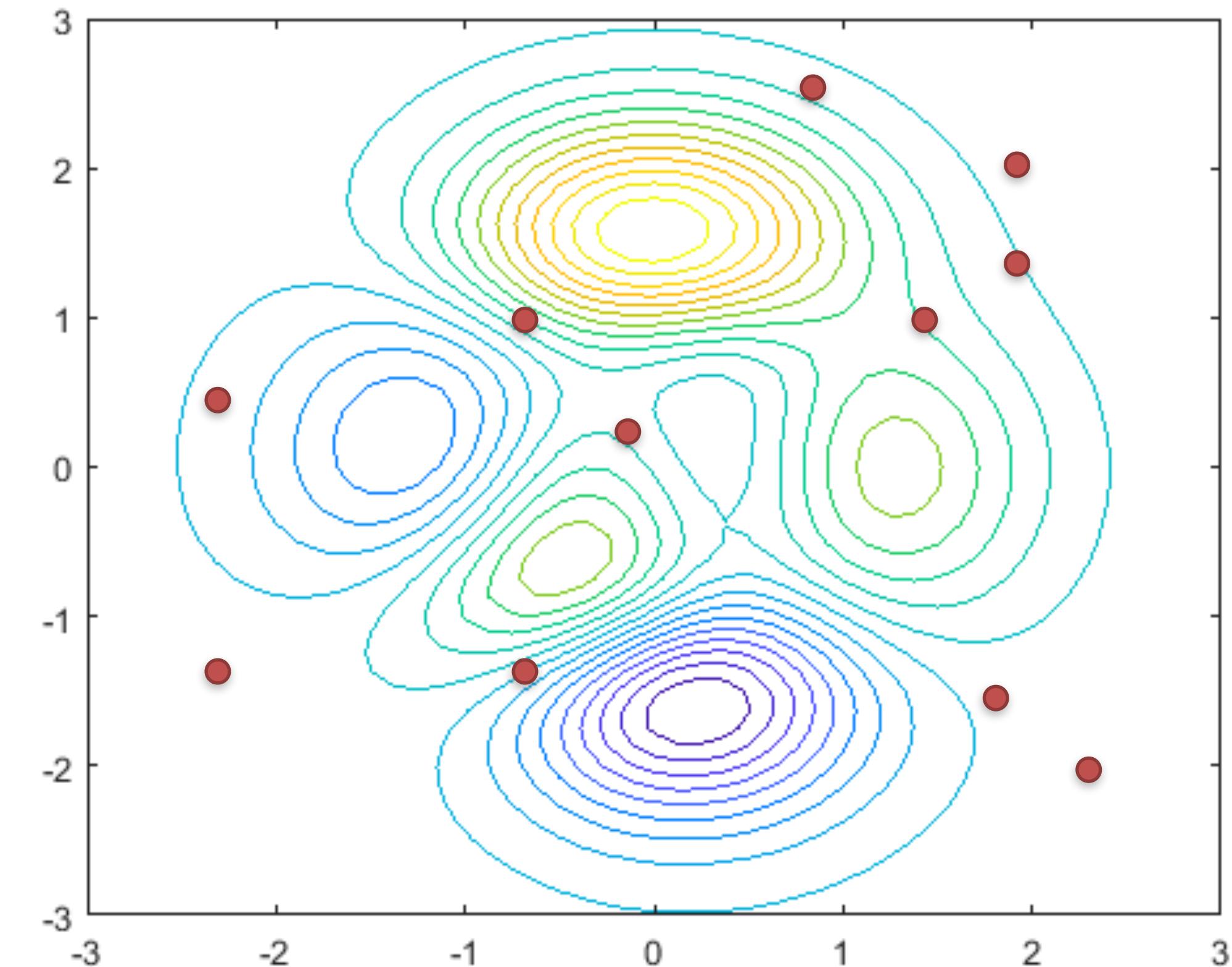
 - `endif`

- `endLoop`



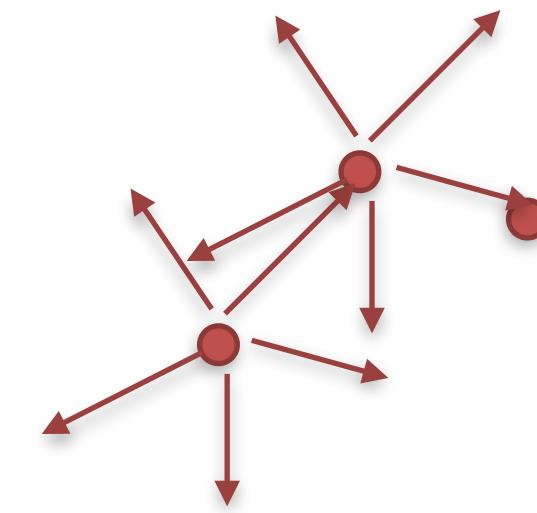
Method 1a: Random search

- `bestLoss = infinity`
- Loop
 - Pick random W
 - Compute loss
 - if $\text{loss} < \text{bestLoss}$
 - $\text{bestLoss} = \text{loss}$
 - $\text{currW} = W$
 - endif
- endLoop



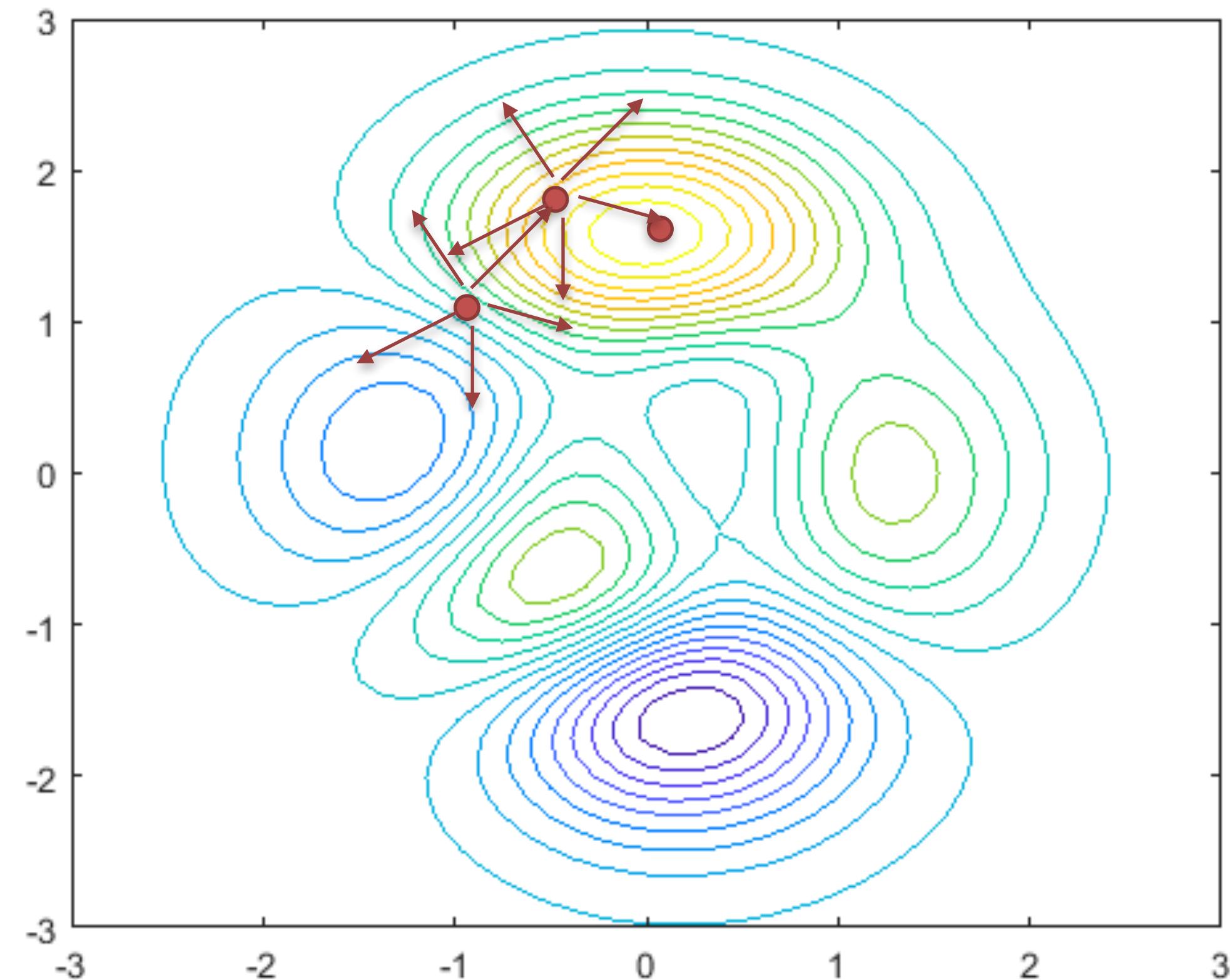
Method 1b: Random search++

- `bestLoss = infinity`
- `W = setRand`
- `stepsize`
- Loop
 - `W_local = W + stepsize * W_random`
 - `Compute loss`
 - `if loss < bestLoss`
 - `bestLoss = loss`
 - `W = W_local`
 - `endif`
- `endLoop`



Method 1b: Random search++

- **bestLoss** = ∞
- **W** = **setRand**
- **stepsize**
- **Loop**
 - W_local** = **W** + **stepsize*****W_random**
 - Compute loss**
 - if** **loss**<**bestLoss**
 - bestLoss** = **loss**
 - W** = **W_local**
 - endif**
- endLoop**



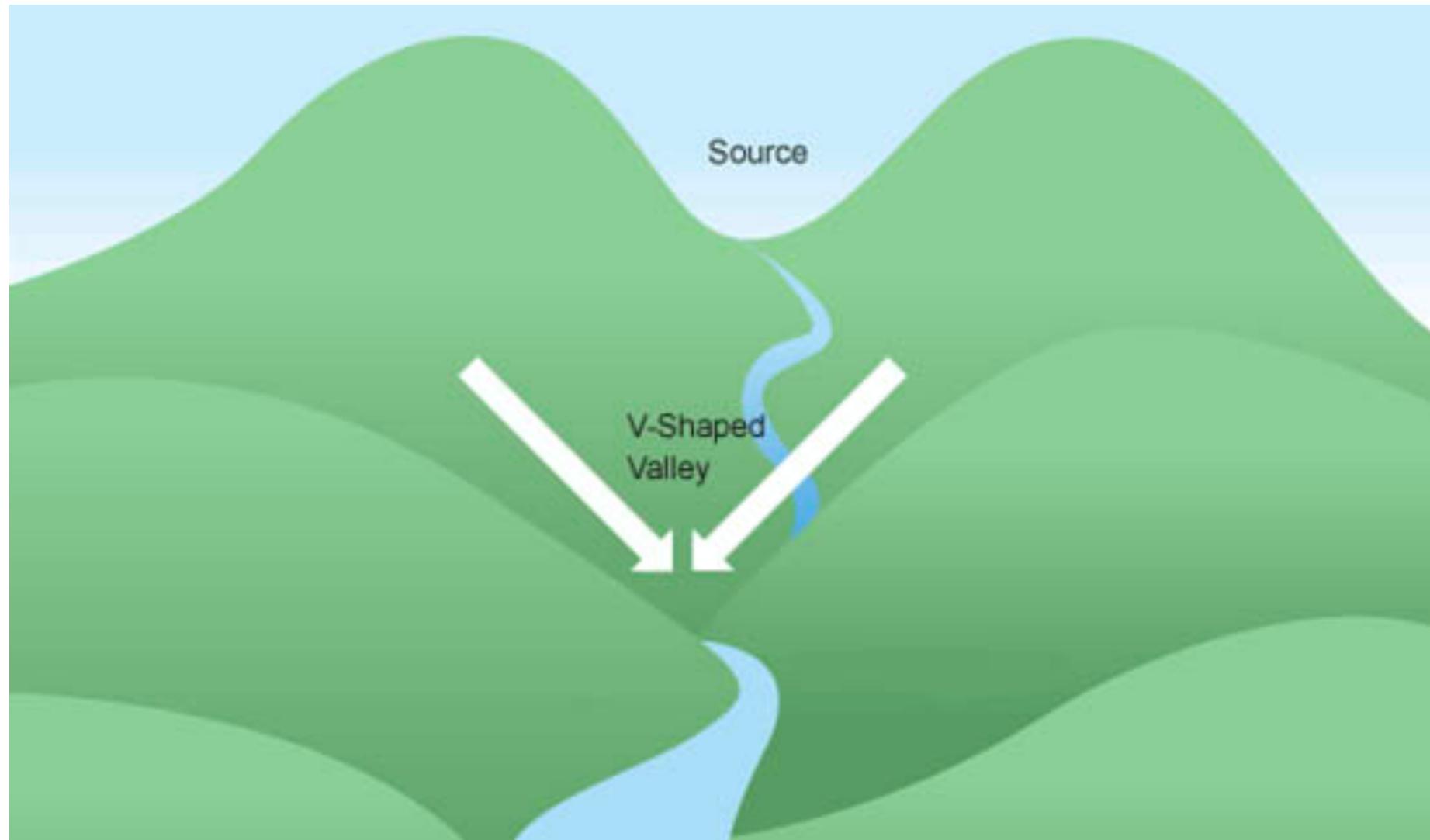
Optimize

$$\mathbf{W}^* = \arg \min g(\mathbf{x}_i, W)$$

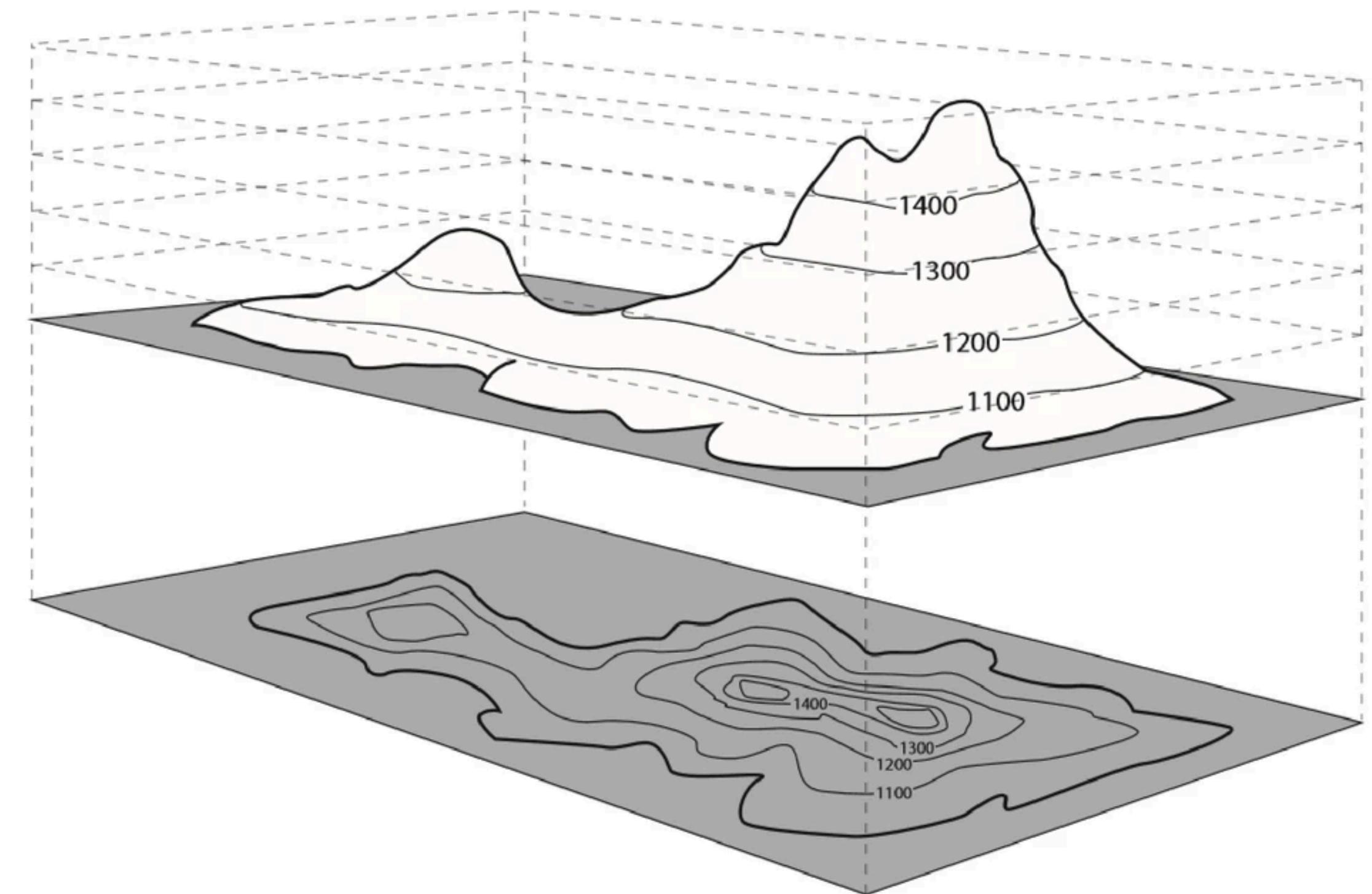
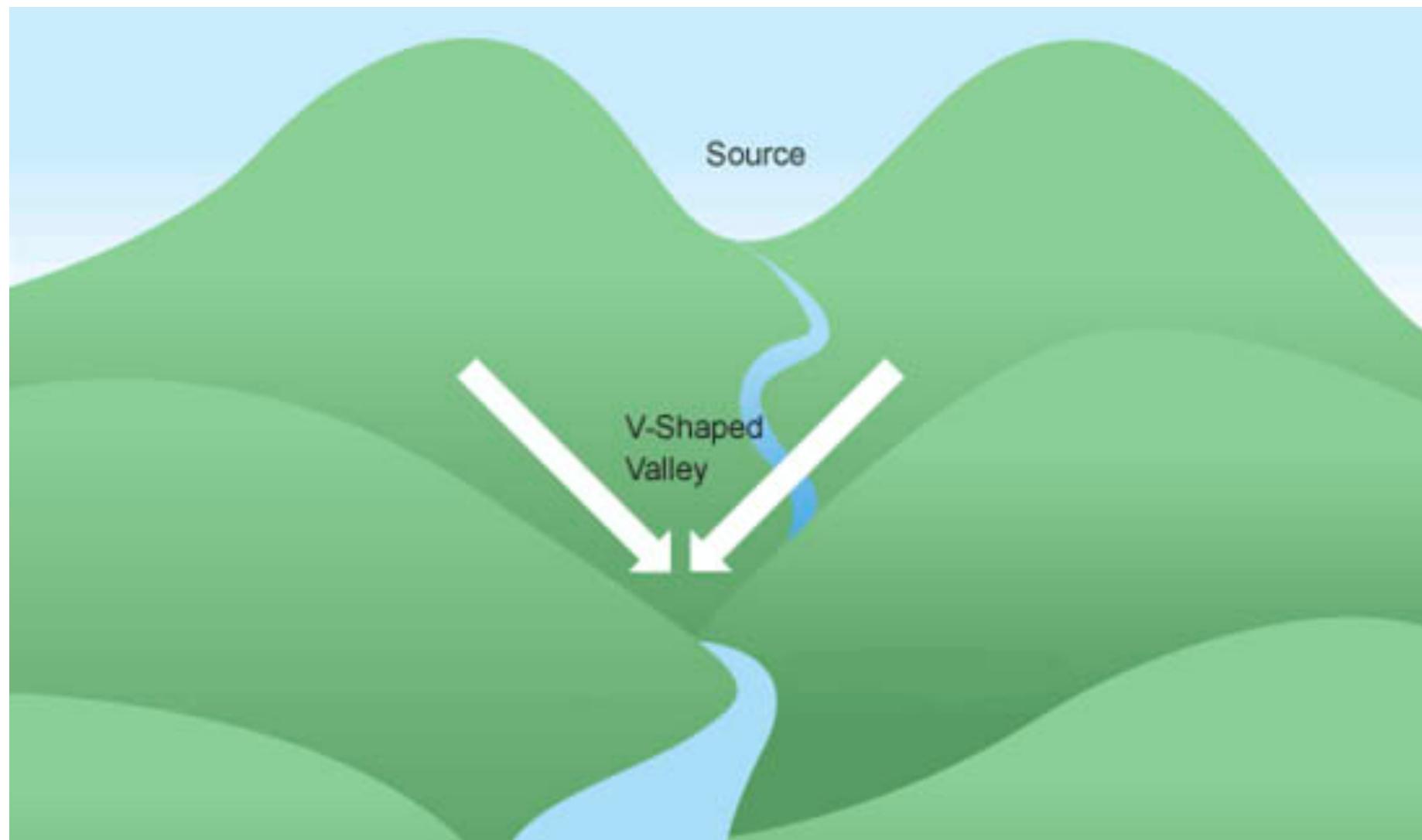
$$\nabla g(\mathbf{x}_i, W) = \left[\frac{\partial g(\mathbf{x}_i, W)}{\partial w_i} = 0 \right]_{D \times 1}$$

Error Landscape

Error Landscape

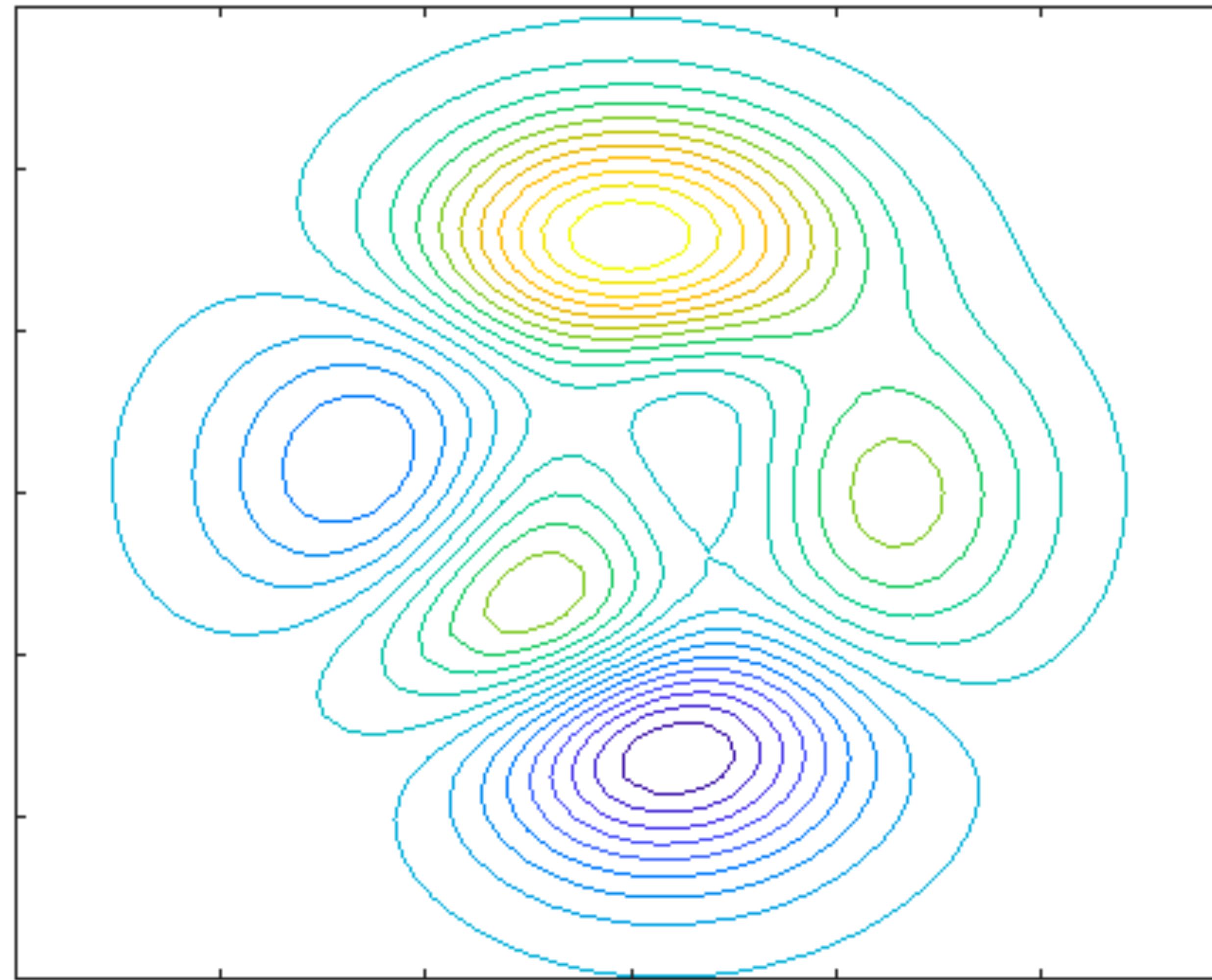


Error Landscape

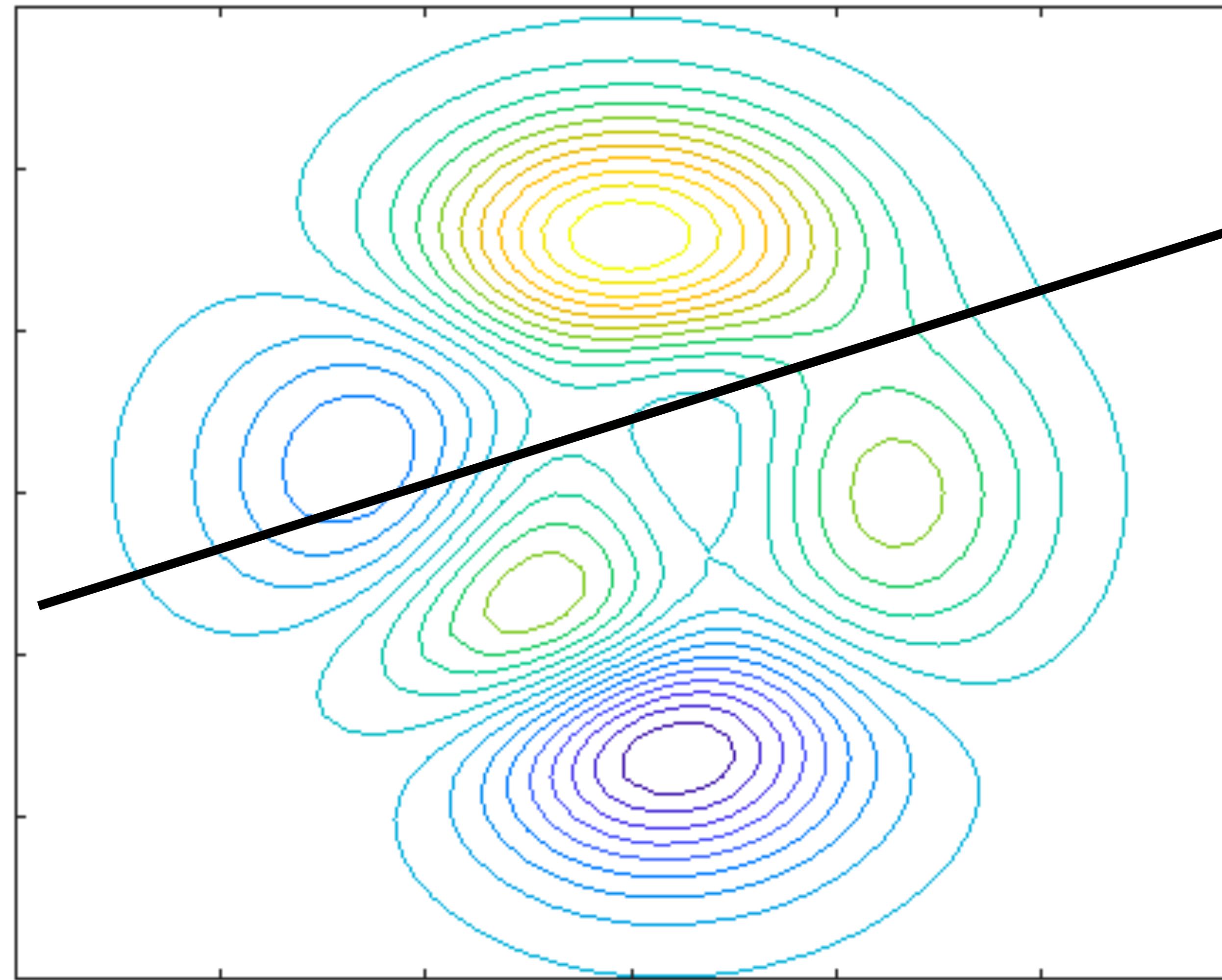


Method 2: Follow Negative of Gradient

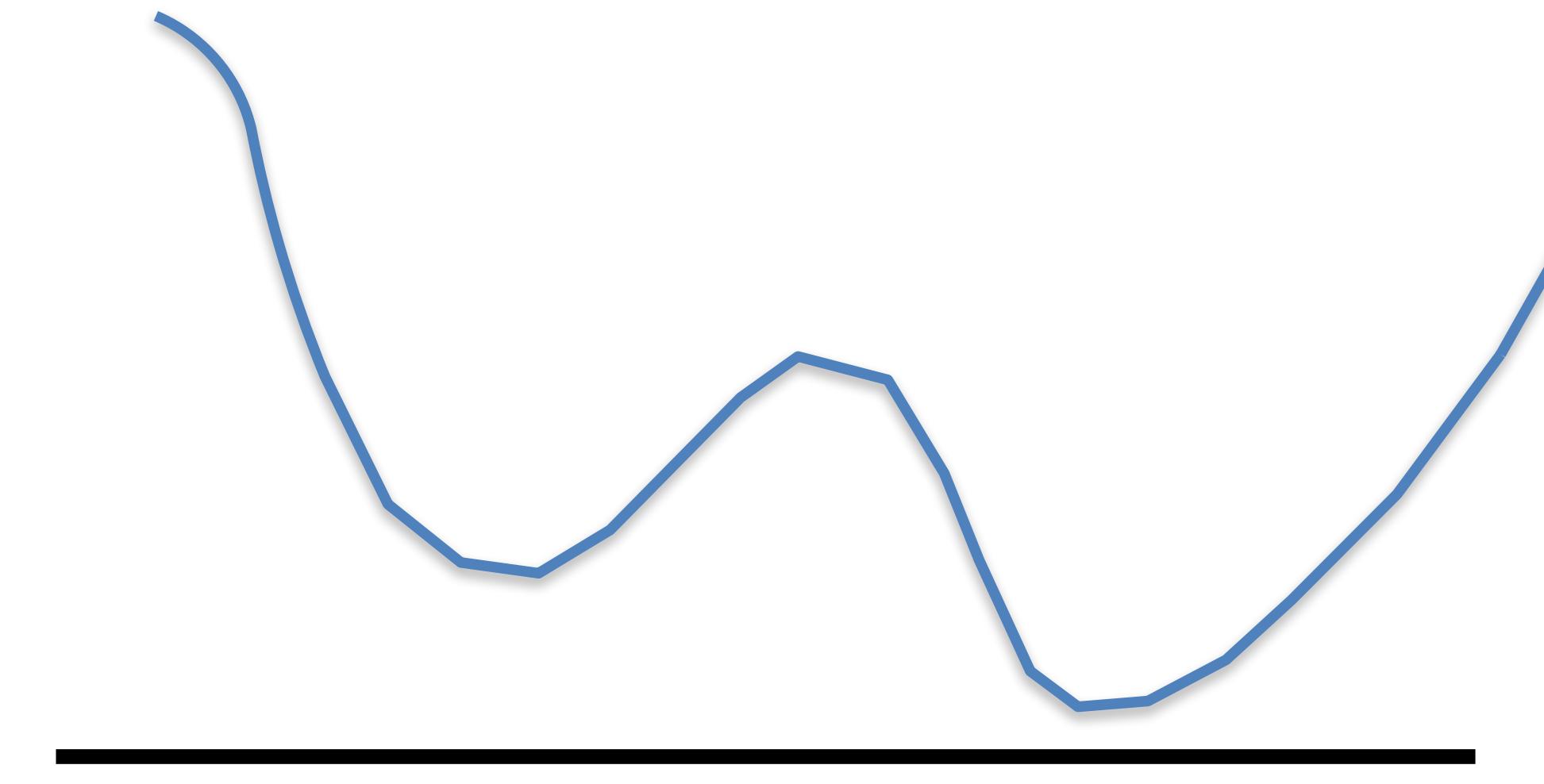
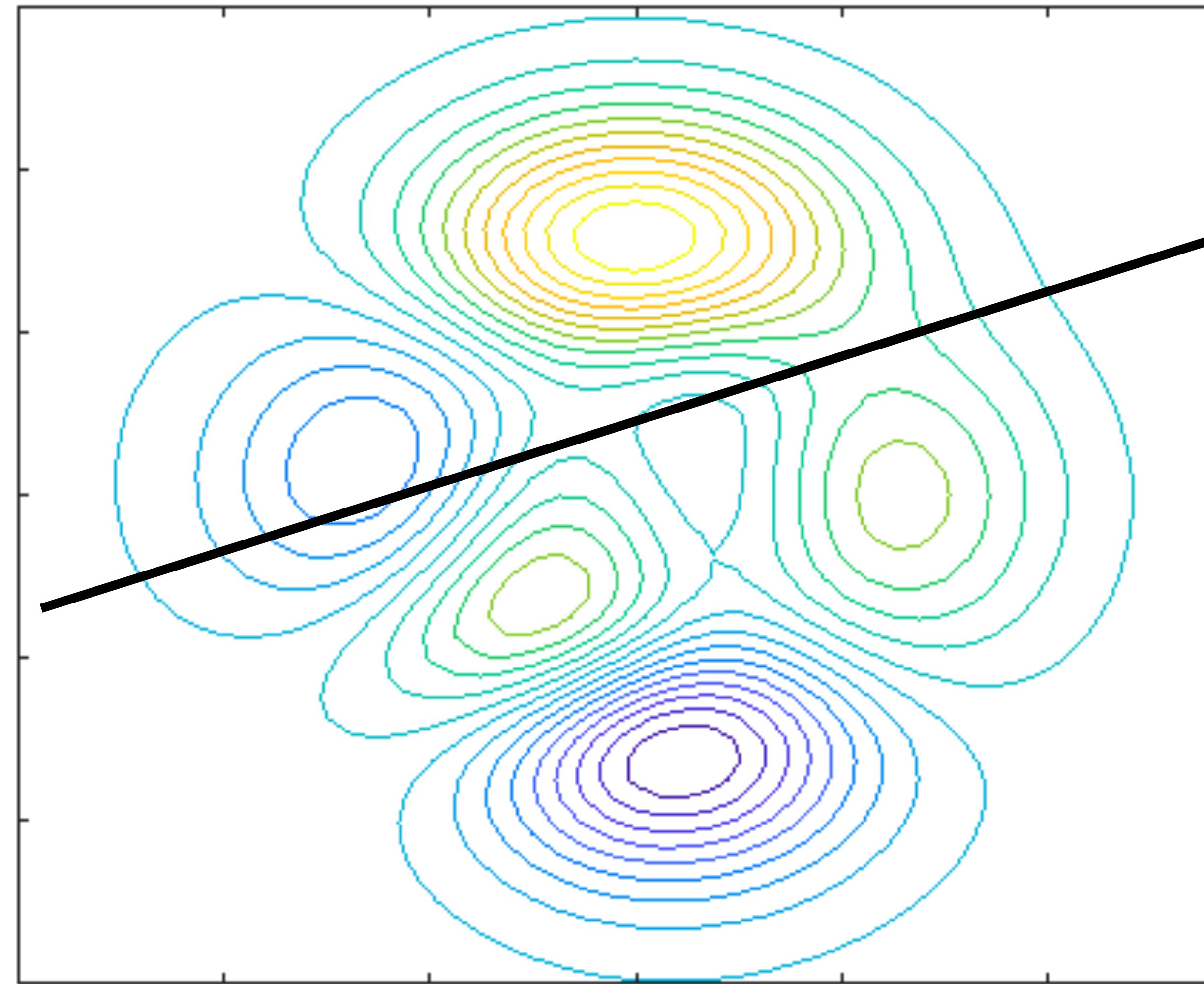
Method 2: Follow Negative of Gradient



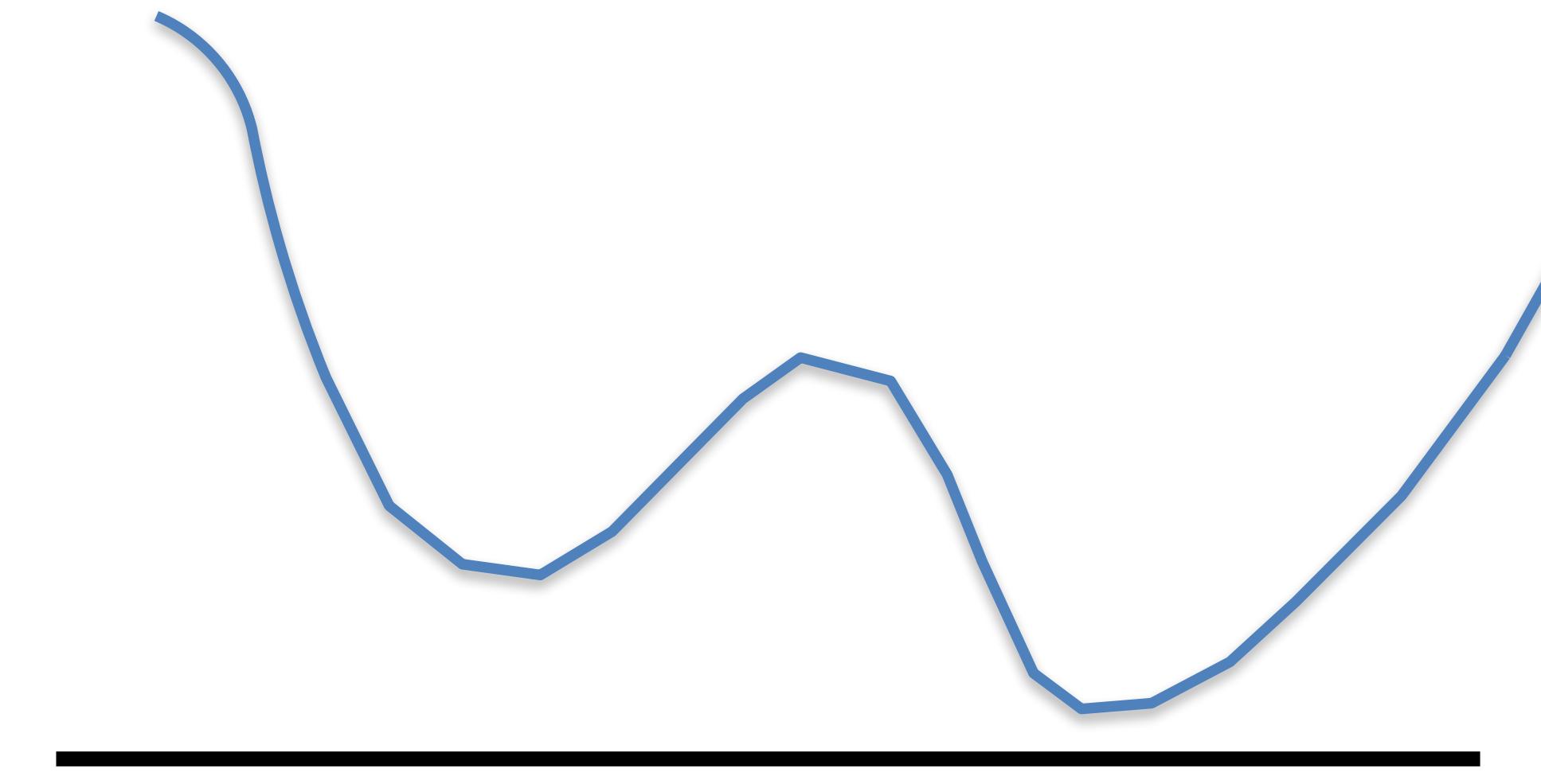
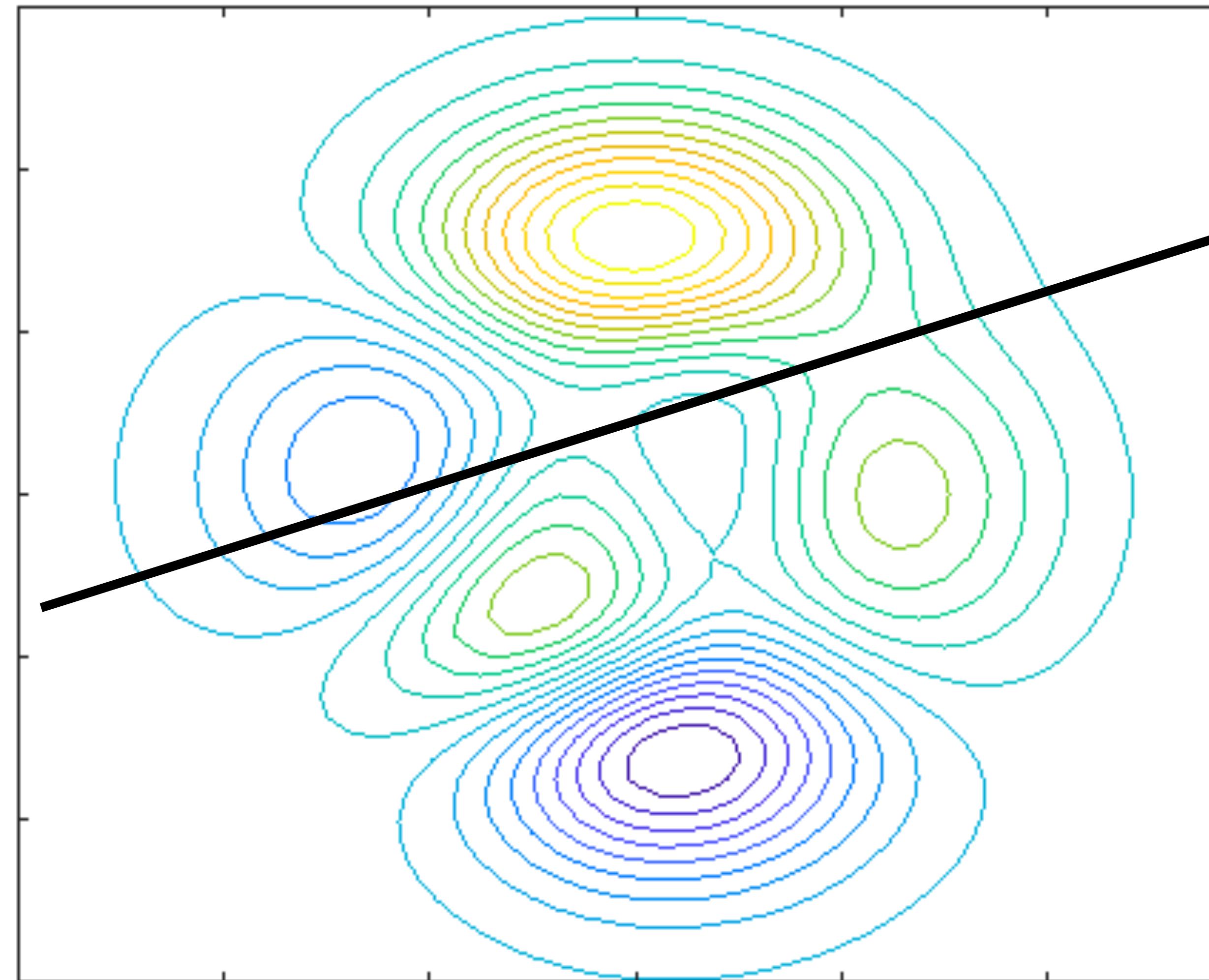
Method 2: Follow Negative of Gradient



Method 2: Follow Negative of Gradient

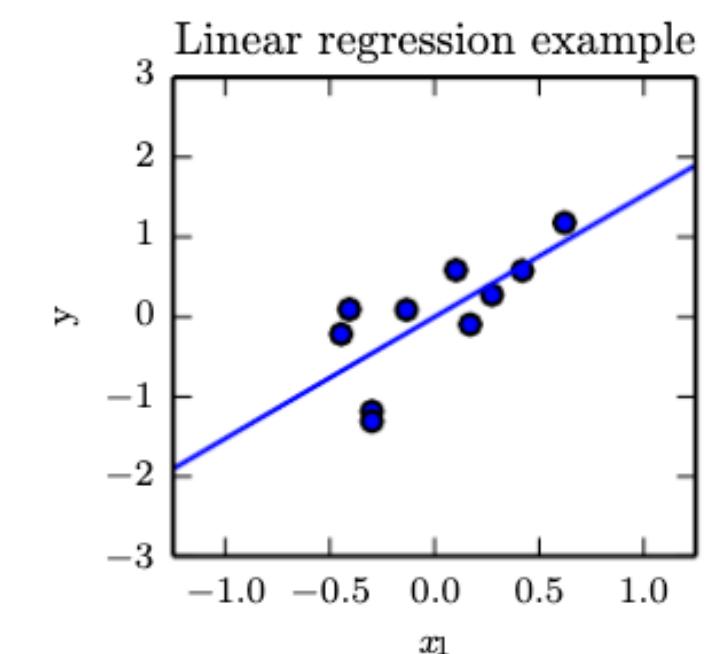


Method 2: Follow Negative of Gradient



$$\frac{df(x)}{dx}$$

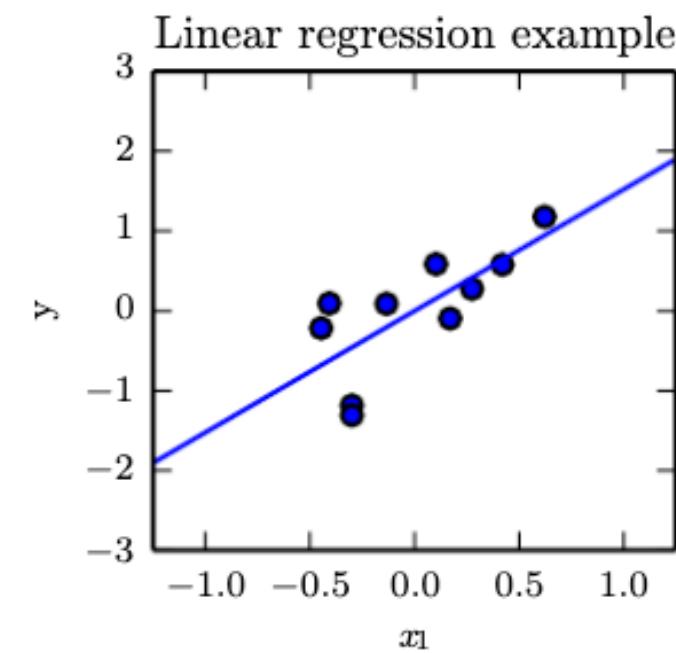
Linear Regression



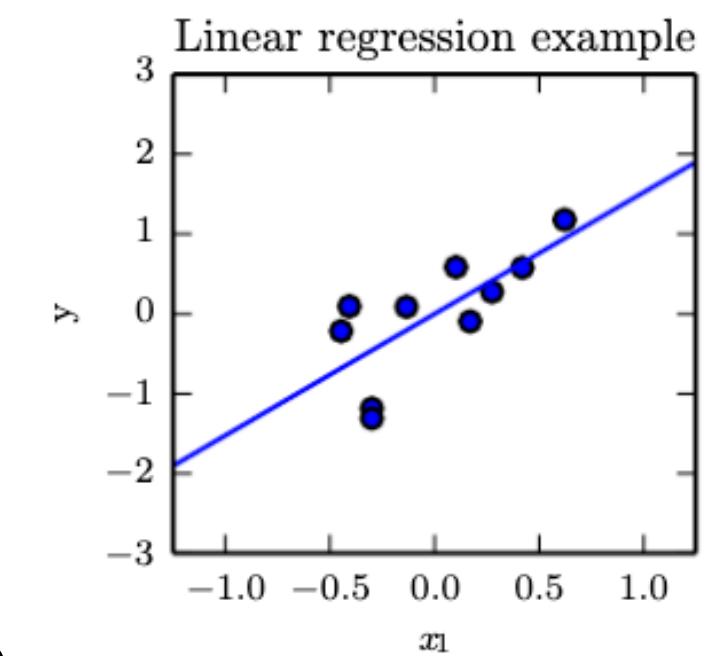
Linear Regression

$$\mathbf{x} \in \mathbb{R}^n; \quad \mathbf{w} \in \mathbb{R}^n; \quad y \in \mathbb{R}$$

$$\hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^T \mathbf{x}$$



Linear Regression

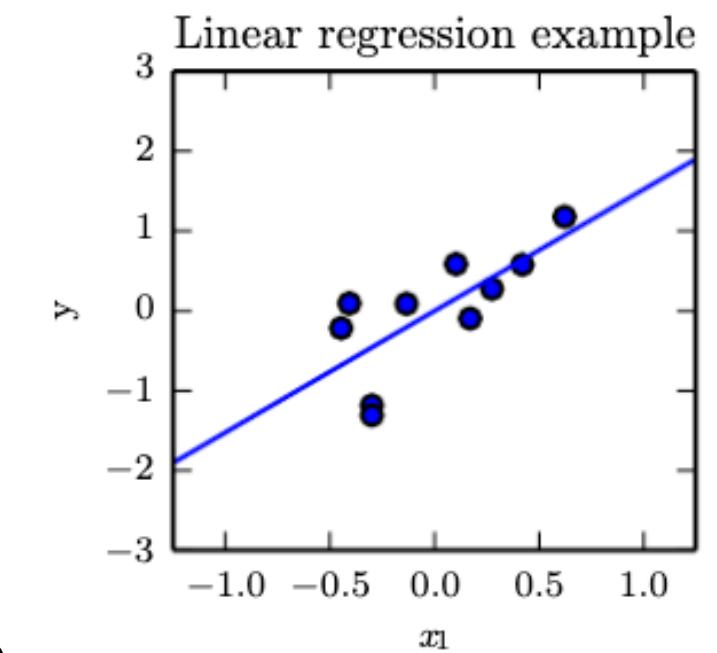


$$\mathbf{x} \in \mathbb{R}^n; \quad \mathbf{w} \in \mathbb{R}^n; \quad y \in \mathbb{R}$$

$$\hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^T \mathbf{x}$$

$$MSE_{\text{test}} = \frac{1}{N} \sum_i (\hat{y}_i^{\text{test}} - y_i^{\text{test}})^2$$

Linear Regression



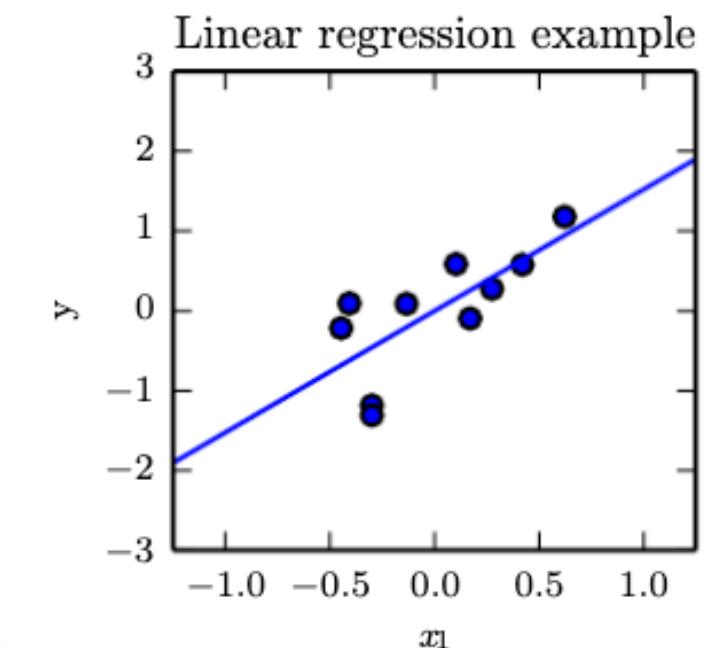
$$\mathbf{x} \in \mathbb{R}^n; \quad \mathbf{w} \in \mathbb{R}^n; \quad y \in \mathbb{R}$$

$$\hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^T \mathbf{x}$$

$$MSE_{\text{test}} = \frac{1}{N} \sum_i (\hat{y}_i^{\text{test}} - y_i^{\text{test}})^2$$

$$\nabla_{\mathbf{w}} MSE_{\text{test}} = 0$$

Linear Regression



$$\mathbf{x} \in \mathbb{R}^n; \quad \mathbf{w} \in \mathbb{R}^n; \quad y \in \mathbb{R}$$

$$\hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^T \mathbf{x}$$

$$MSE_{\text{test}} = \frac{1}{N} \sum_i (\hat{y}_i^{\text{test}} - y_i^{\text{test}})^2$$

$$\nabla_{\mathbf{w}} MSE_{\text{test}} = 0$$

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

Linear Regression

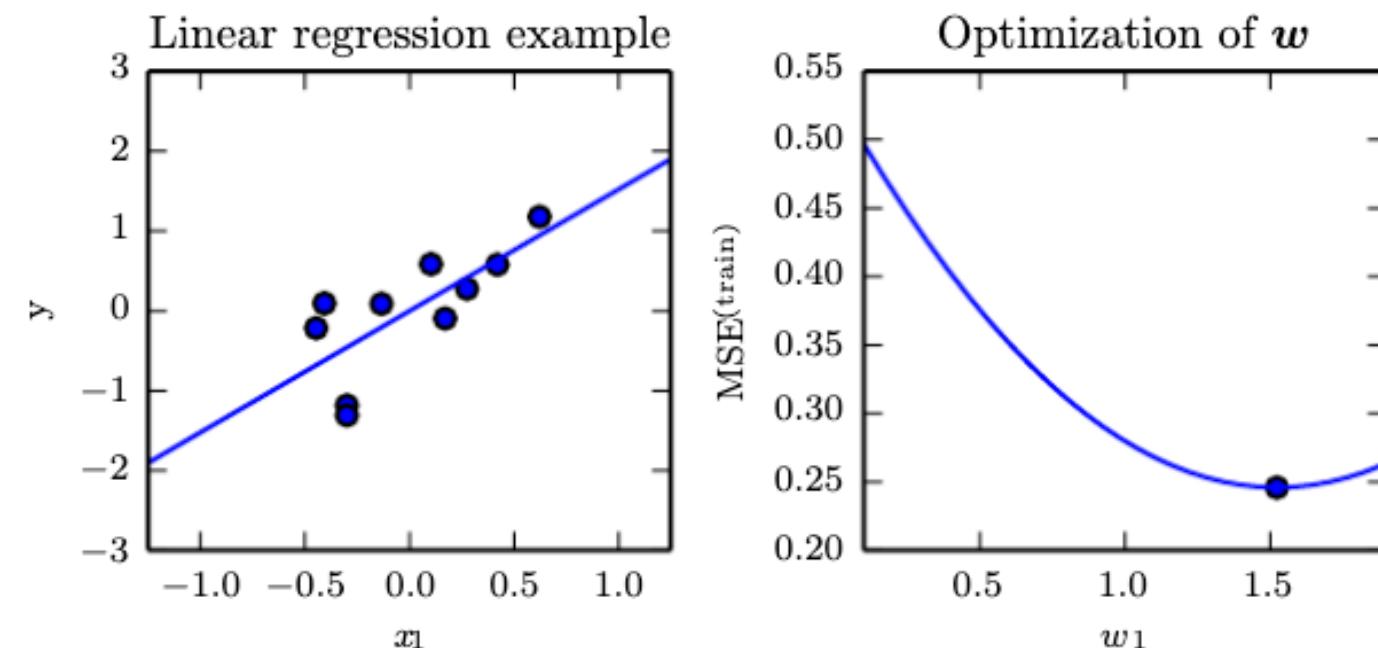
$$\mathbf{x} \in \mathbb{R}^n; \quad \mathbf{w} \in \mathbb{R}^n; \quad y \in \mathbb{R}$$

$$\hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^T \mathbf{x}$$

$$MSE_{\text{test}} = \frac{1}{N} \sum_i (\hat{y}_i^{\text{test}} - y_i^{\text{test}})^2$$

$$\nabla_{\mathbf{w}} MSE_{\text{test}} = 0$$

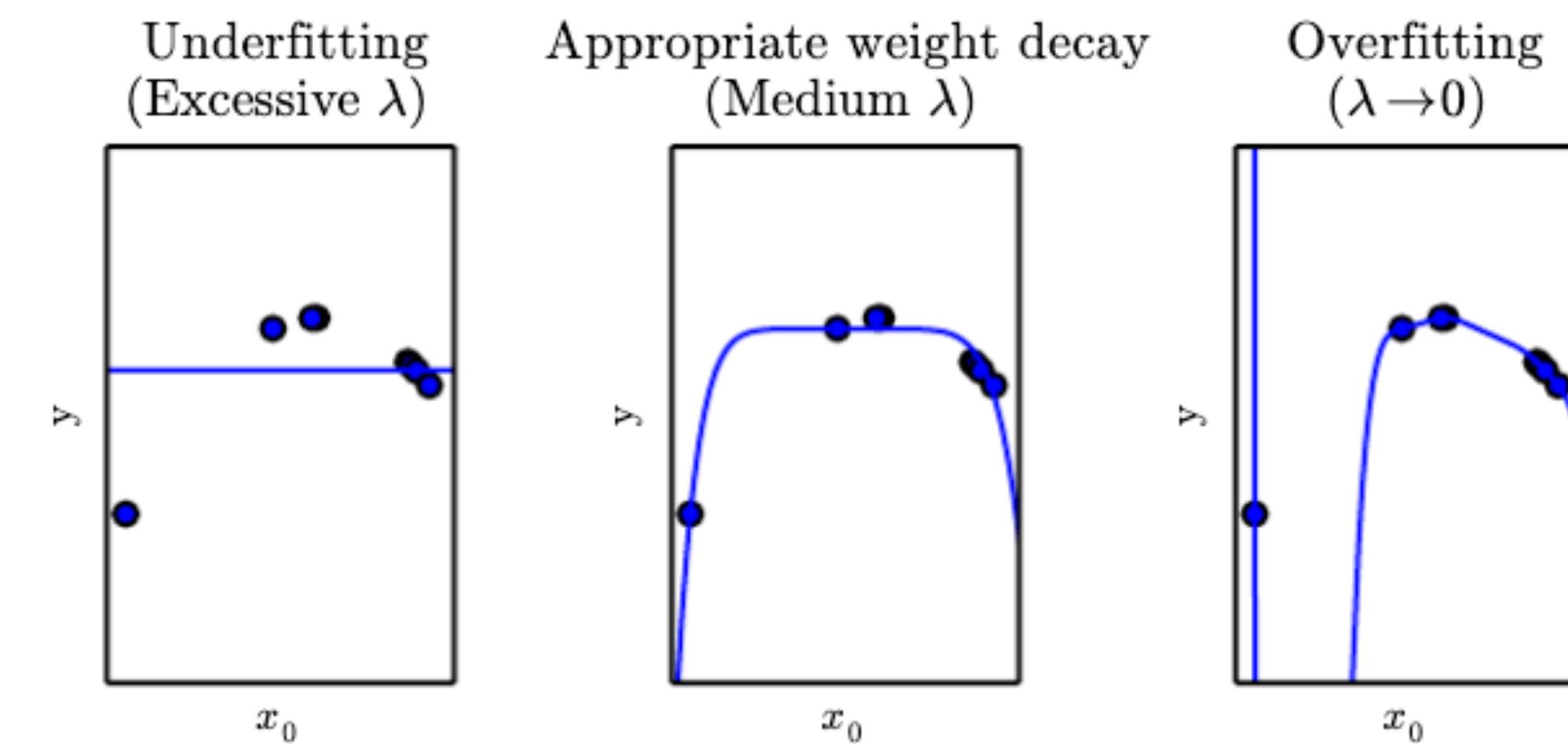
$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$



Polynomial Regression

- What happens if we allow higher powers

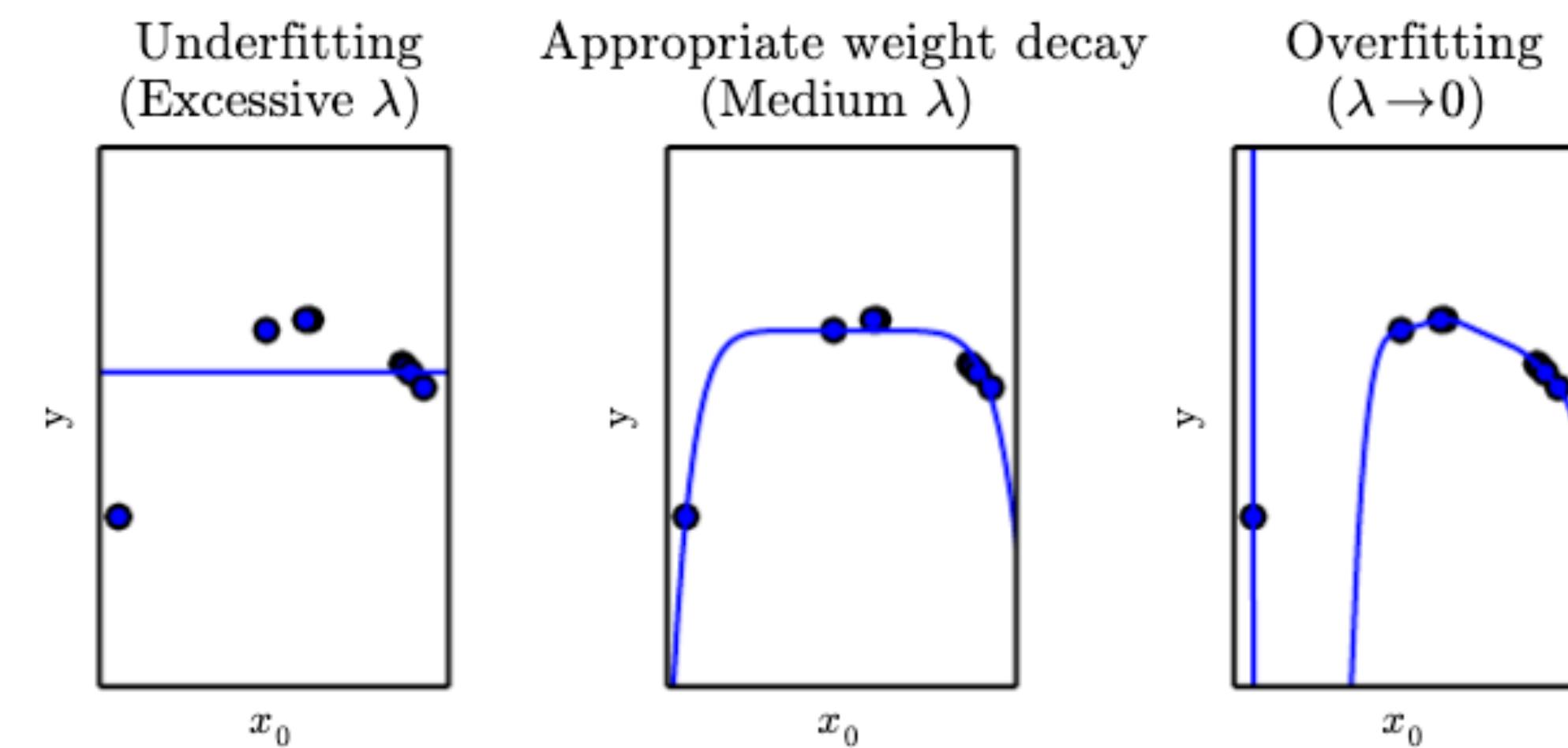
- We need additional regular



Polynomial Regression

- What happens if we allow higher powers

- We need additional regular

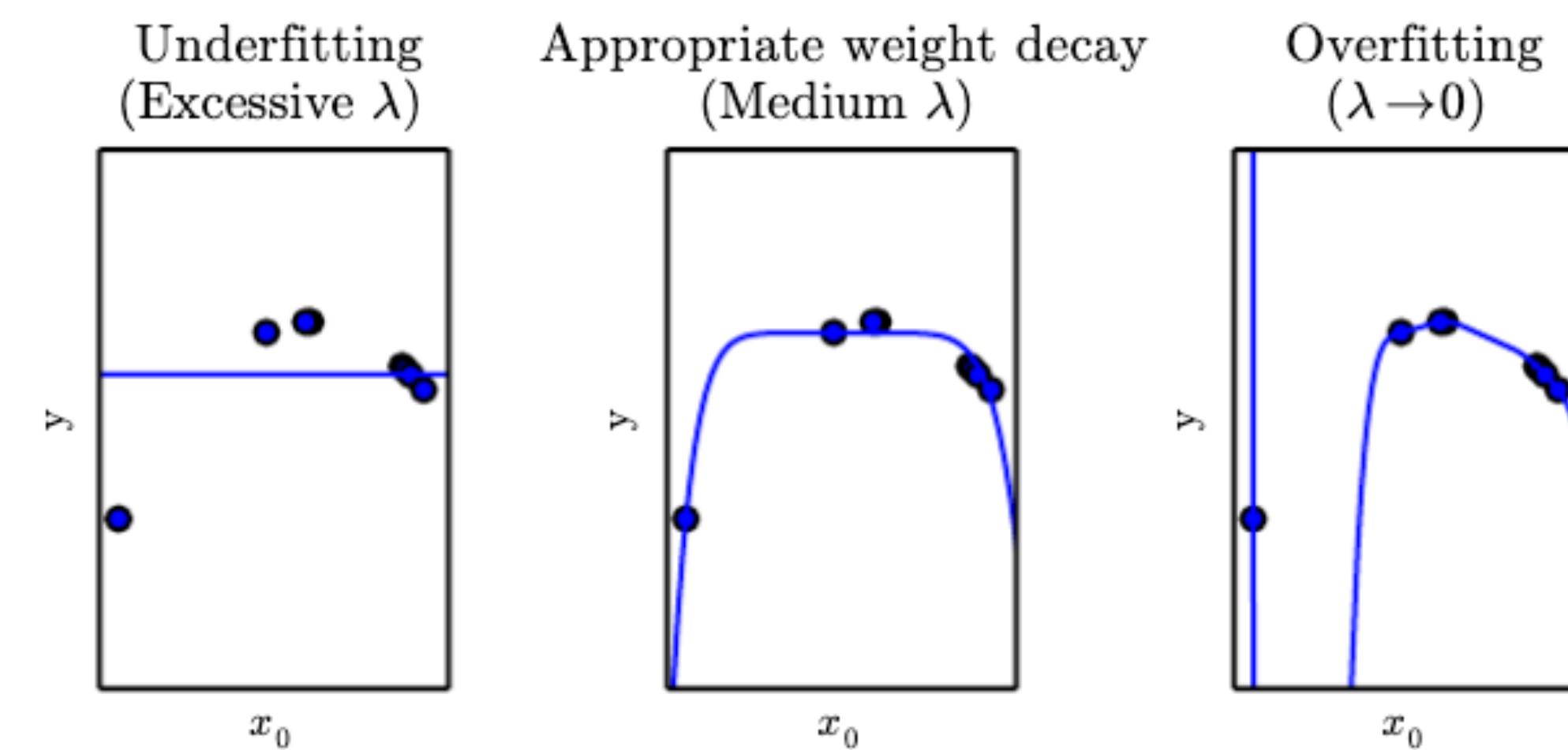


$$MSE_{\text{test}} = \frac{1}{N} \sum_i (\hat{y}_i^{\text{test}} - y_i^{\text{test}})^2 + \lambda \mathbf{w}^t \mathbf{w}$$

Polynomial Regression

- What happens if we allow higher powers

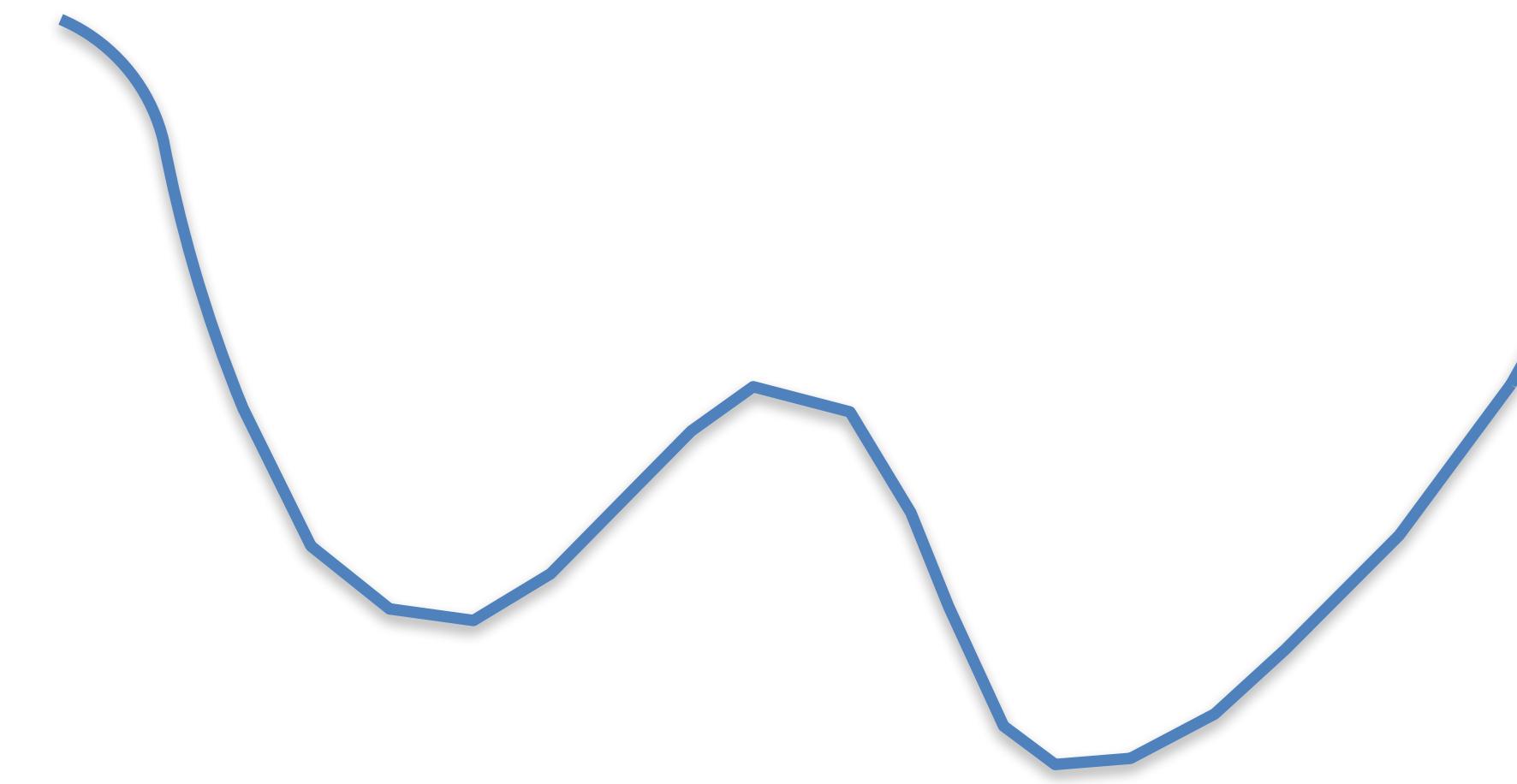
- We need additional regular



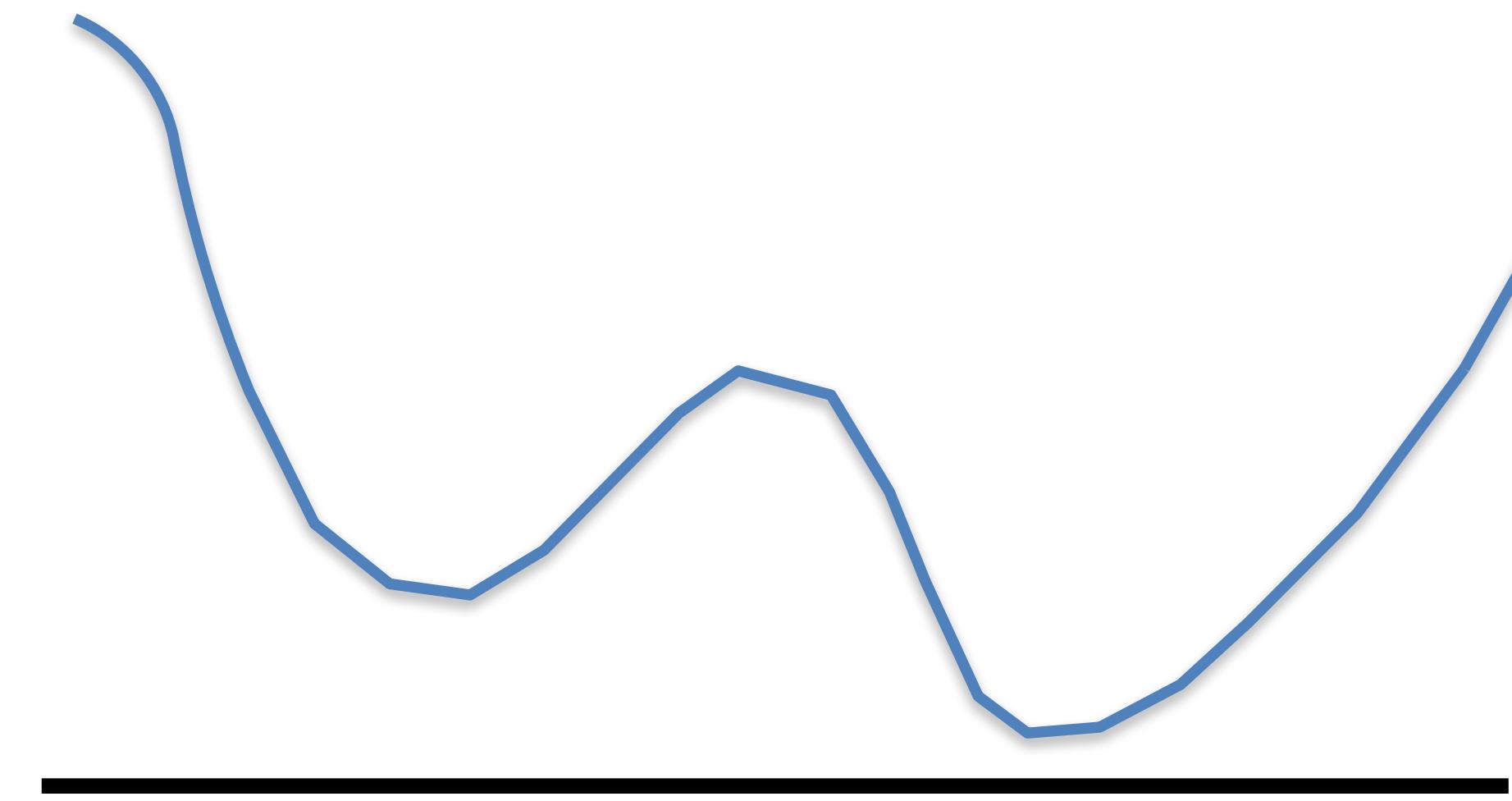
$$MSE_{\text{test}} = \frac{1}{N} \sum_i (\hat{y}_i^{\text{test}} - y_i^{\text{test}})^2 + \boxed{\lambda \mathbf{w}^t \mathbf{w}}$$

Regularizer

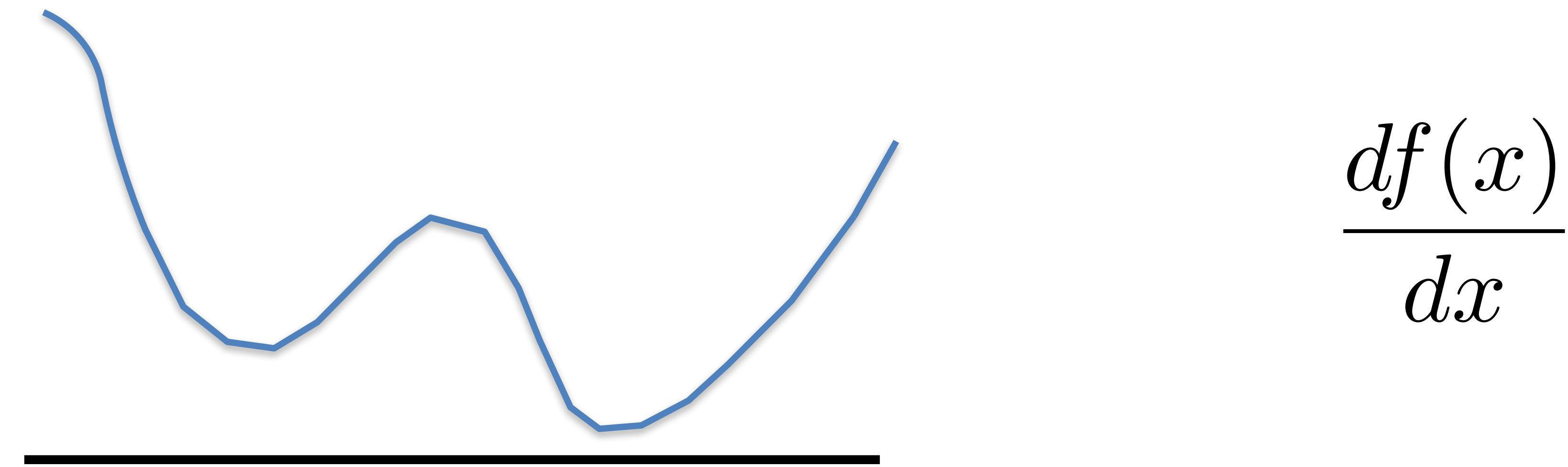
Method 2: Follow Negative of Gradient



Method 2: Follow Negative of Gradient



Method 2: Follow Negative of Gradient



Compute $f(x)$, numerically

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Stochastic Gradient Descent

Stochastic Gradient Descent

$$J(\mathbf{w}) = \mathbb{E}_{\mathbf{x}, y \sim p_{\text{data}}} L(\mathbf{x}, y, \mathbf{w}) = \frac{1}{N} \sum_i L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

Stochastic Gradient Descent

$$J(\mathbf{w}) = \mathbb{E}_{\mathbf{x}, y \sim p_{\text{data}}} L(\mathbf{x}, y, \mathbf{w}) = \frac{1}{N} \sum_i L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{N} \sum_i \nabla_{\mathbf{w}} L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

Stochastic Gradient Descent

$$J(\mathbf{w}) = \mathbb{E}_{\mathbf{x}, y \sim p_{\text{data}}} L(\mathbf{x}, y, \mathbf{w}) = \frac{1}{N} \sum_i L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{N} \sum_i \nabla_{\mathbf{w}} L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) \approx \frac{1}{m} \sum_i \nabla_{\mathbf{w}} L(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w})$$

Compute $f(x)$, numerically

0.45
1.34
3.26
-1.45
3.2

0.45+0
1.34+0
3.26+0.0001
-1.45+0
3.2+0

Numerical vs Analytical Gradients

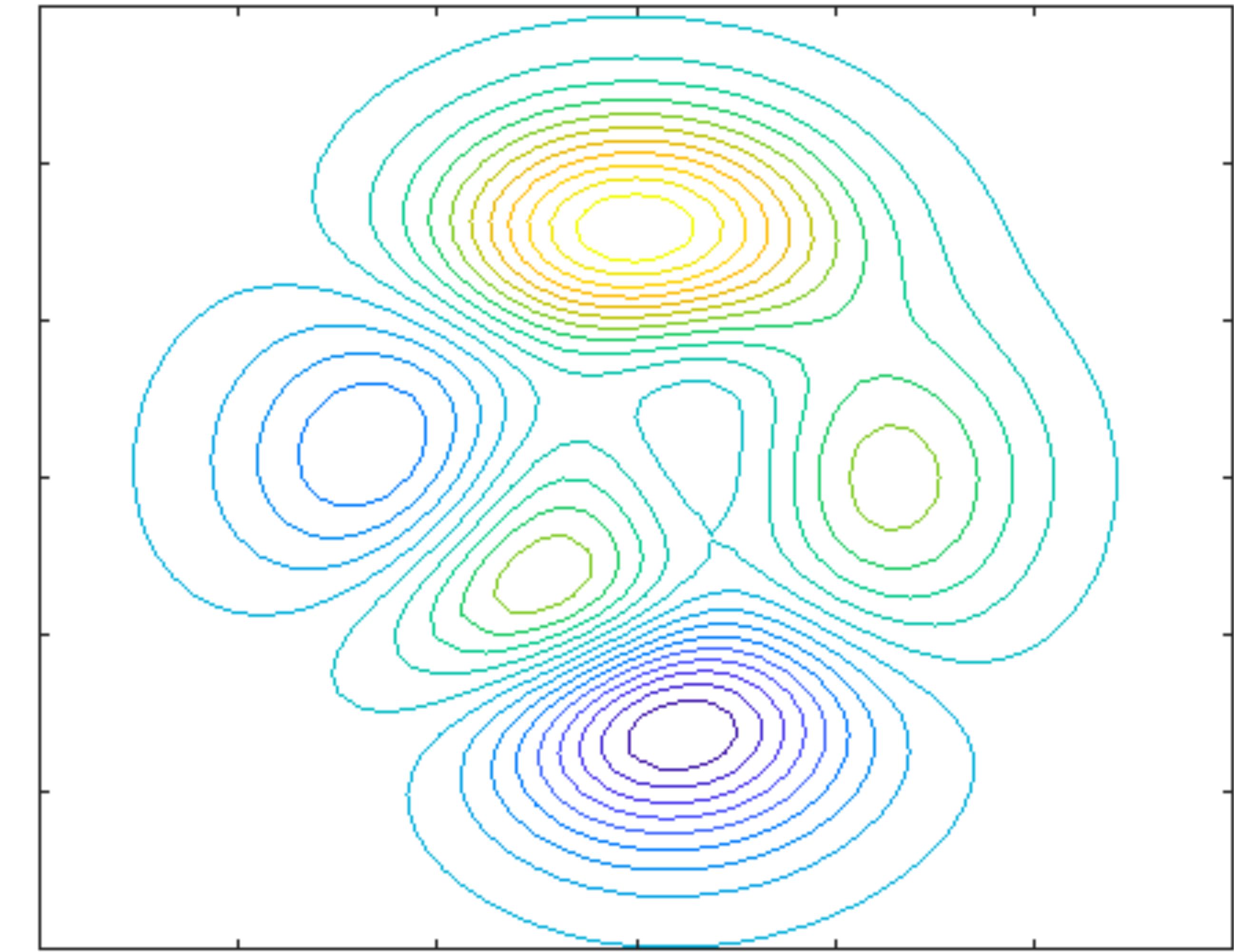
- **Slow versus Fast**
- **Easy versus error-prone**
- **Check using numerical gradients**

Gradient Descent

$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

Gradient Descent

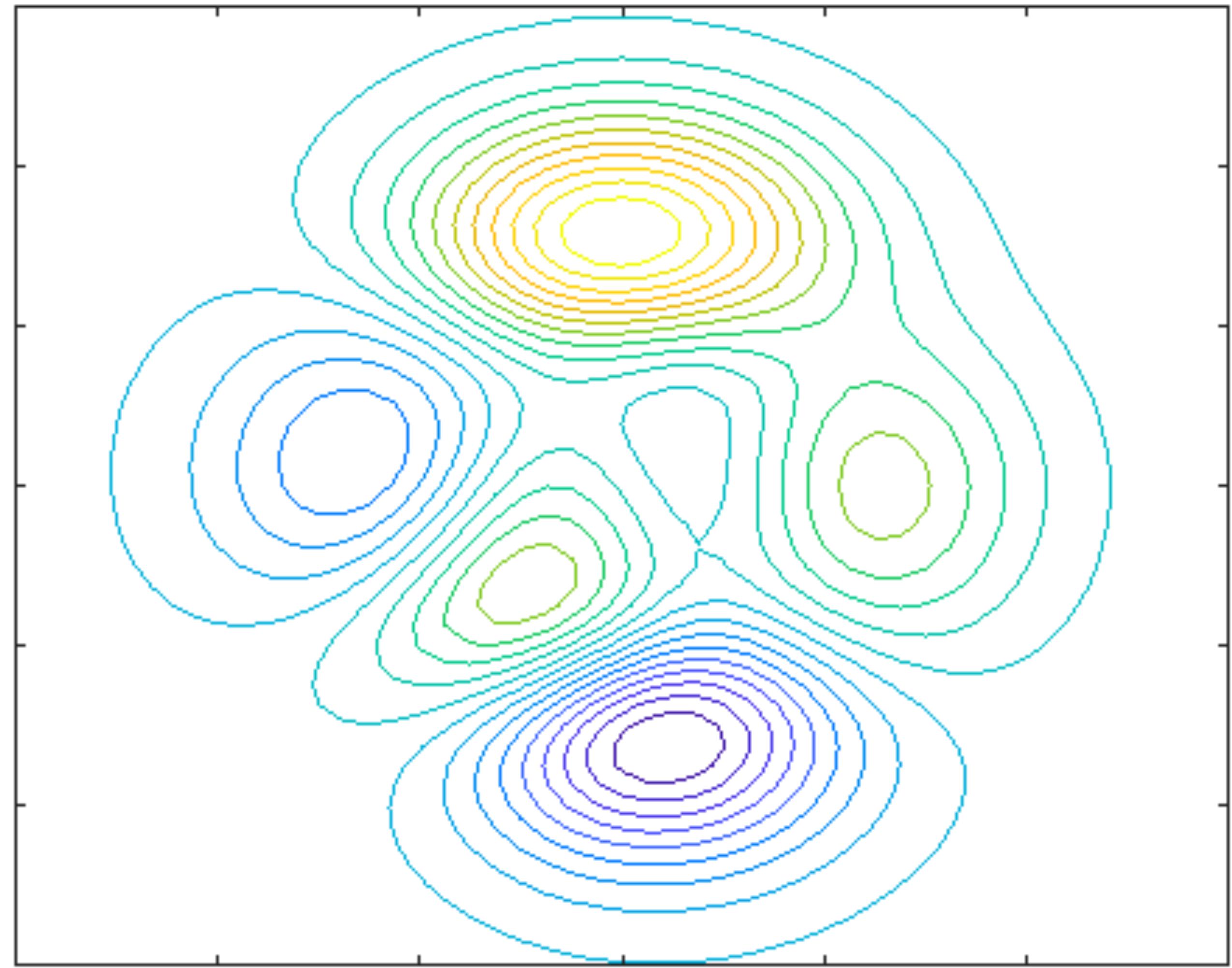
$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$



Gradient Descent

$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

```
w = initialize_weights()
for t in range(num_steps):
    dw = compute_gradient(loss_fn, data, w)
    w -= learning_rate * dw
```

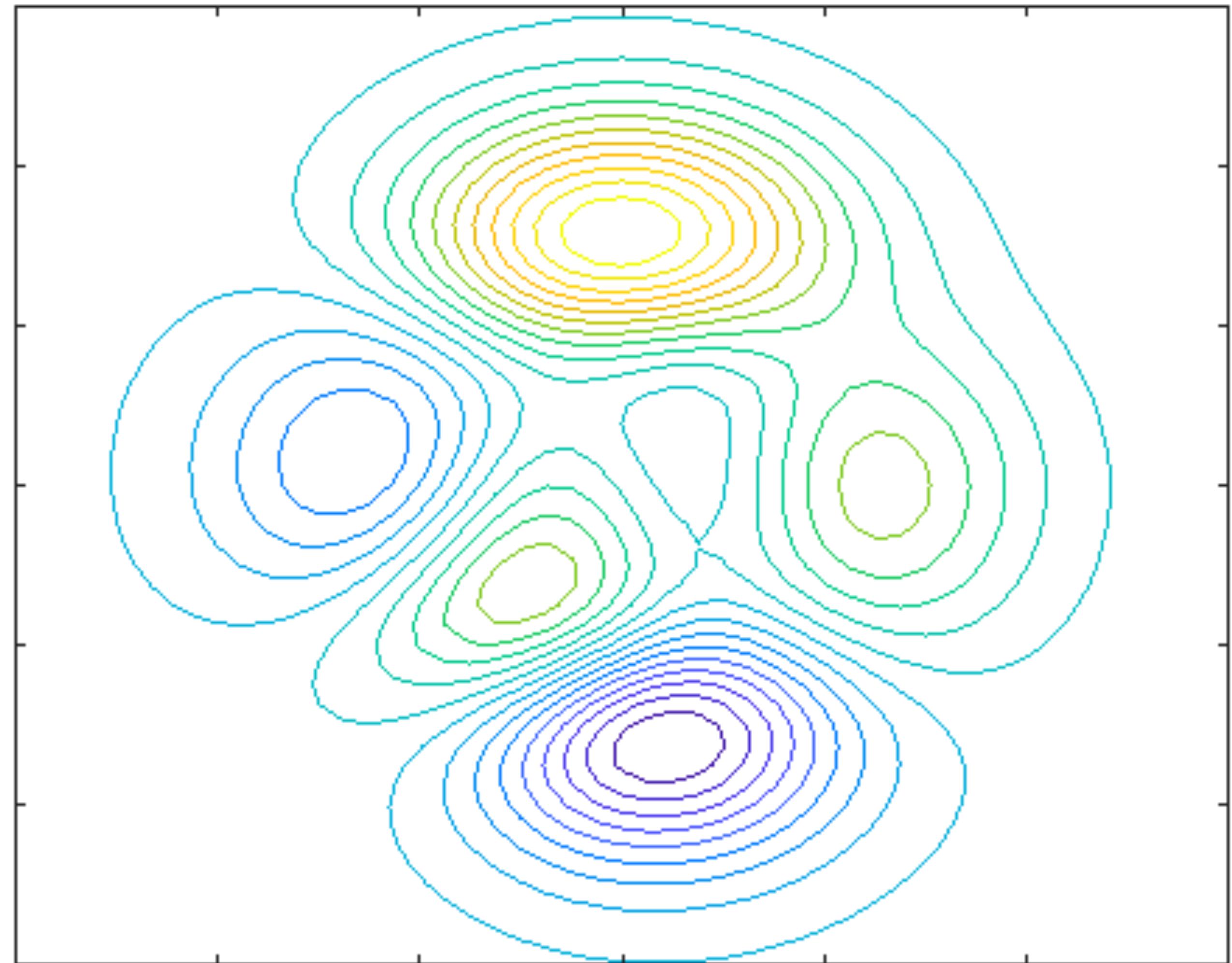


Gradient Descent

$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

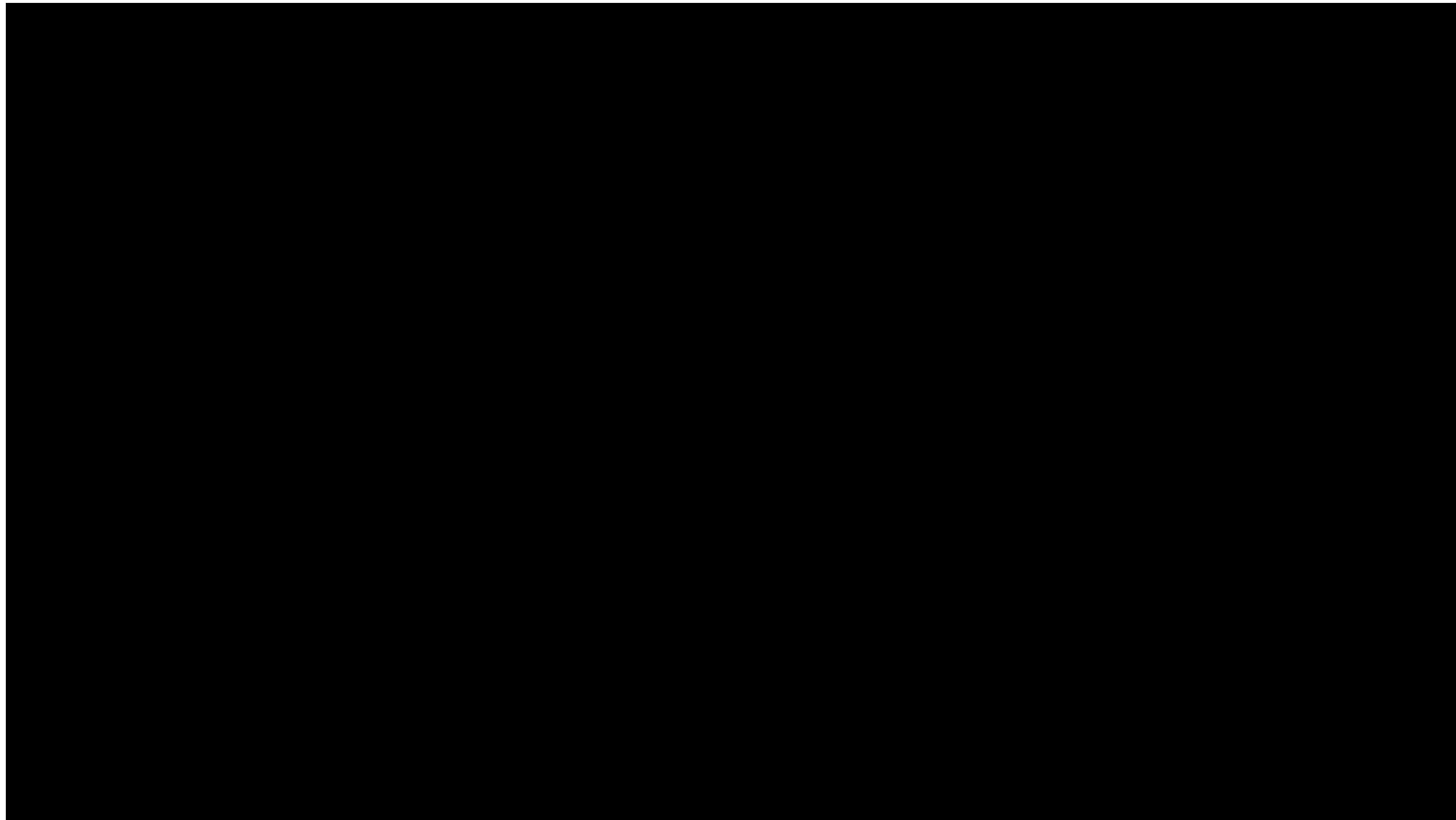
```
w = initialize_weights()  
for t in range(num_steps):  
    dw = compute_gradient(loss_fn, data, w)  
    w -= learning_rate * dw
```

Hyperparameters?



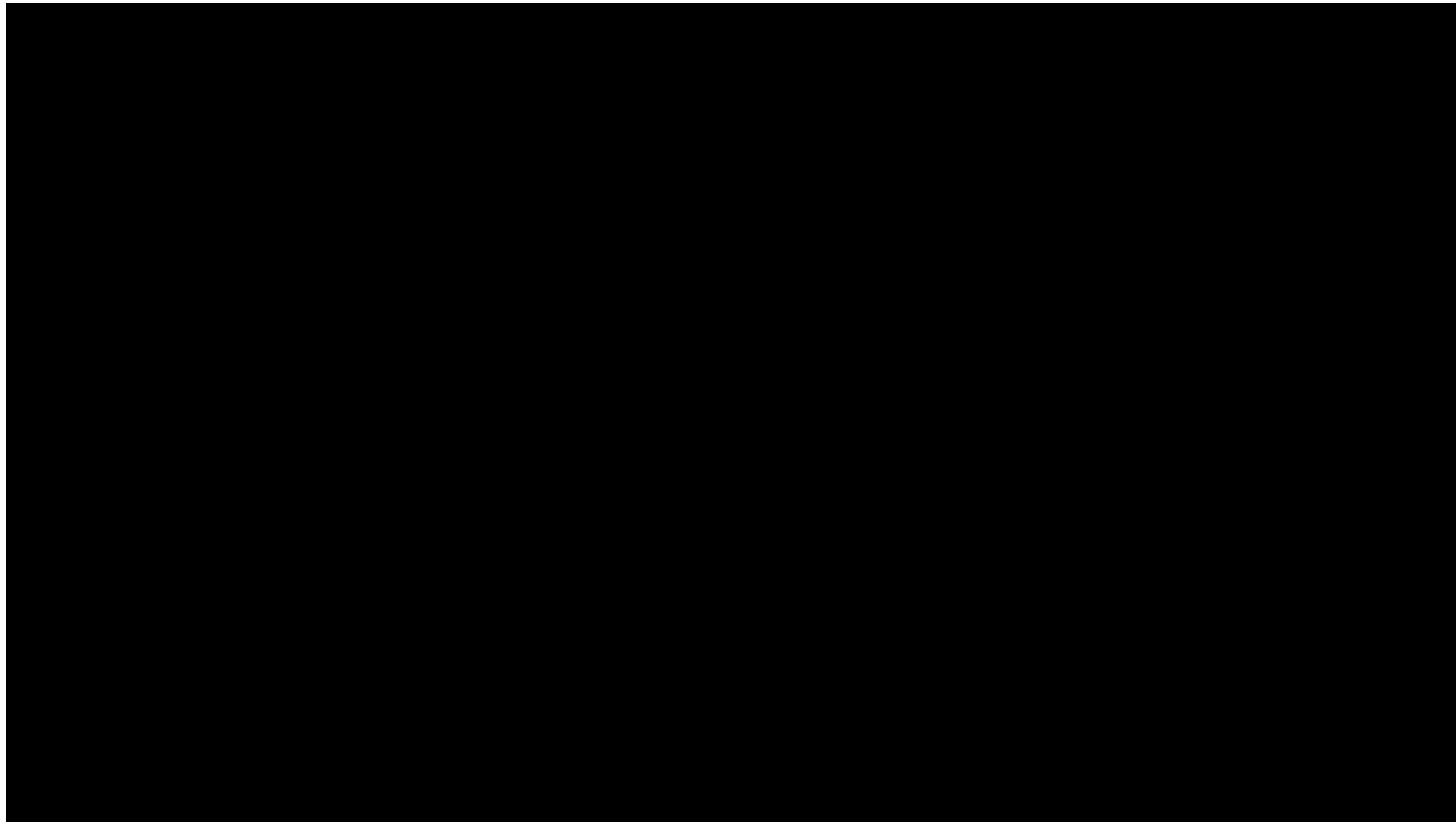
Negative LogLikelihood

$$\log \mathbb{P}(\mathbf{D}|\theta) = \sum_i (y_i \log \hat{y}_{\theta,i} + (1 - y_i) \log(1 - \hat{y}_{\theta,i}))$$



Negative LogLikelihood

$$\log \mathbb{P}(\mathbf{D}|\theta) = \sum_i (y_i \log \hat{y}_{\theta,i} + (1 - y_i) \log(1 - \hat{y}_{\theta,i}))$$



Batch Gradient Descent

Stochastic Gradient Descent (SGD)

$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

Stochastic Gradient Descent (SGD)

$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

```
w = initialize_weights()
for t in range(num_steps):
    minibatch = sample_data(data, batch_size)
    dw = compute_gradient(loss_fn, minibatch, w)
    w -= learning_rate * dw
```

Interactive Demo

Linear Classification Loss Visualization

These linear classifiers were written in Javascript for Stanford's [CS231n: Convolutional Neural Networks for Visual Recognition](#).

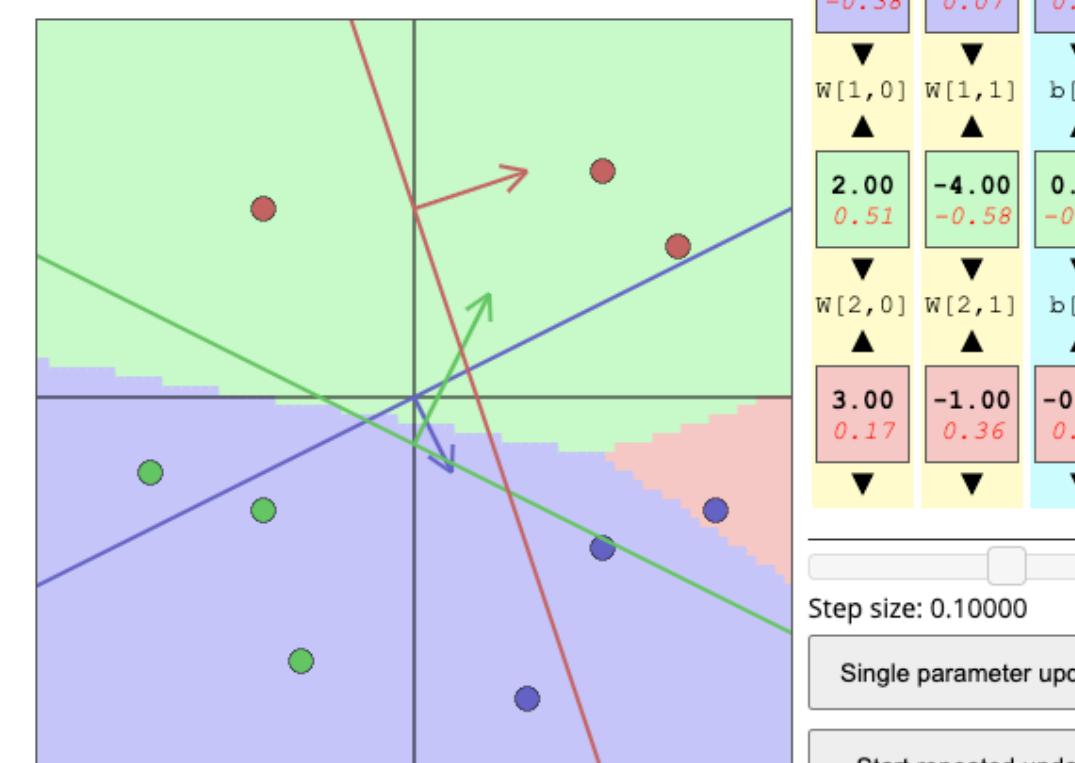
The class scores for linear classifiers are computed as $f(x_i; W, b) = Wx_i + b$, where the parameters consist of weights W and biases b . The training data is x_i with labels y_i . In this demo, the datapoints x_i are 2-dimensional and there are 3 classes, so the weight matrix is of size [3 x 2] and the bias vector is of size [3 x 1]. The multiclass loss function can be formulated in many ways. The default in this demo is an SVM that follows [Weston and Watkins 1999]. Denoting f as the [3 x 1] vector that holds the class scores, the loss has the form:

$$L = \underbrace{\frac{1}{N} \sum_i \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)}_{\text{data loss}} + \lambda \underbrace{\sum_k \sum_l W_{k,l}^2}_{\text{regularization loss}}$$

Where N is the number of examples, and λ is a hyperparameter that controls the strength of the L2 regularization penalty $R(W) = \sum_k \sum_l W_{k,l}^2$. On the bottom right of this demo you can also flip to different formulations for the Multiclass SVM including One vs All (OVA) where a separate binary SVM is trained for every class independently (vs. other classes all labeled as negatives), and Structured SVM which maximizes the margin between the correct score and the score of the highest runner-up class. You can also choose to use the cross-entropy loss which is used by the Softmax classifier. These losses are explained in the [CS231n notes on Linear Classification](#).

Datapoints are shown as circles colored by their class. Parameters W, b are shown (red/green/blue). The background regions are colored by below. The value is in bold whenever class is most likely at any point according to the current weights. Each classifier is visualized by a line that indicates its zero score level set. For example, the blue *italic* below. Click the classifier computes scores as $W_{0,0}x_0 + W_{0,1}x_1 + b_0$ and the triangles to control the blue line shows the set of points (x_0, x_1) that give score of zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is.

Note: you can drag the datapoints.



Visualization of the data loss computation. Each row is loss due to one datapoint. The first three columns are the 2D data x_i and the label y_i . The next three columns are the three class scores from each classifier $f(x_i; W, b) = Wx_i + b$ (E.g. $s[0] = x[0] * W[0,0] + x[1] * W[0,1] + b[0]$). The last column is the data loss for a single example, L_i .

$x[0]$	$x[1]$	y	$s[0]$	$s[1]$	$s[2]$	L
0.50	0.40	0	1.30	-0.10	0.60	0.30
0.80	0.30	0	1.40	0.90	1.60	1.70
0.30	0.80	0	1.90	-2.10	-0.40	0.00
-0.40	0.30	1	0.20	-1.50	-2.00	3.20
-0.30	0.70	1	1.10	-2.90	-2.10	6.80
-0.70	0.20	1	-0.30	-1.70	-2.80	2.40
0.70	-0.40	2	-0.10	3.50	2.00	2.50
0.50	-0.60	2	-0.70	3.90	1.60	3.30
-0.40	-0.50	2	-1.40	1.70	-1.20	4.70
mean:						2.77

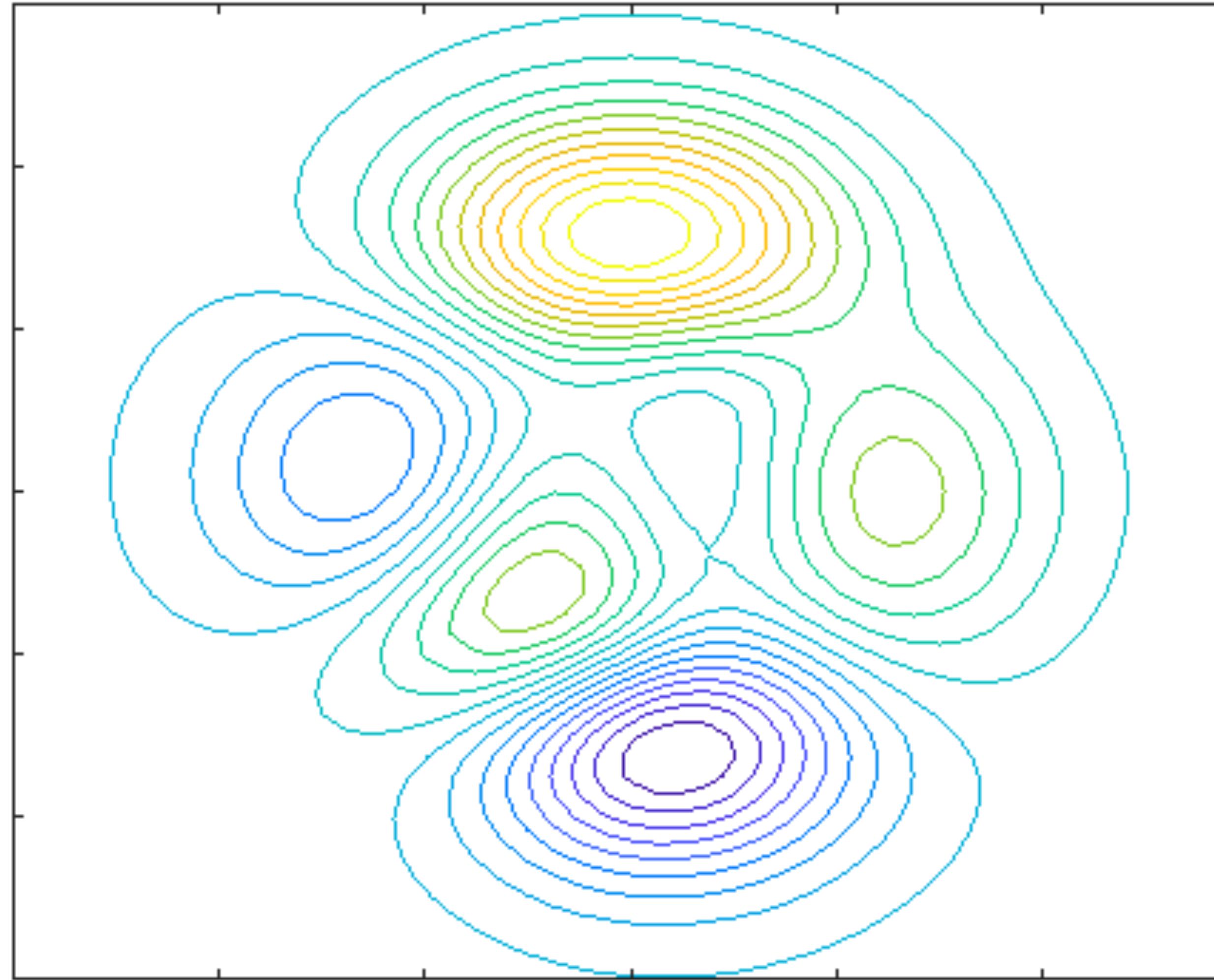
Total data loss: 2.77
Regularization loss: 3.50
Total loss: 6.27

L2 Regularization strength: 0.10000

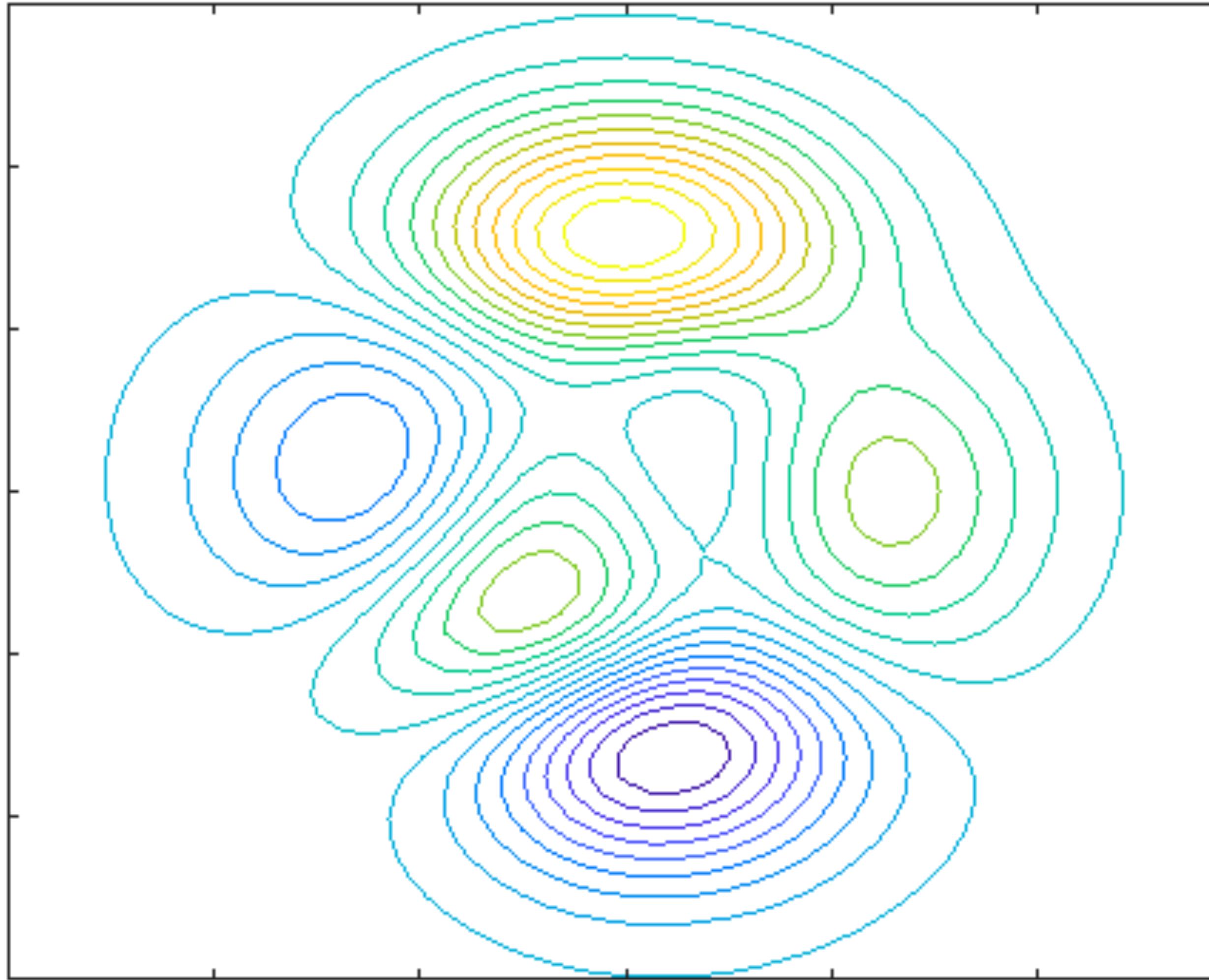
Multiclass SVM loss formulation:
 Weston Watkins 1999
 One vs. All
 Structured SVM
 Softmax

Problems with GD (or SGD)

Problems with GD (or SGD)



Problems with GD (or SGD)



$$v_{t+1} = \rho v_t + \nabla f(w_t)$$
$$w_{t+1} = w_t - \alpha v_{t+1}$$

Form of posterior distribution

Bernoulli-type conditional distribution

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

Particular choice of form of f:

Form of posterior distribution

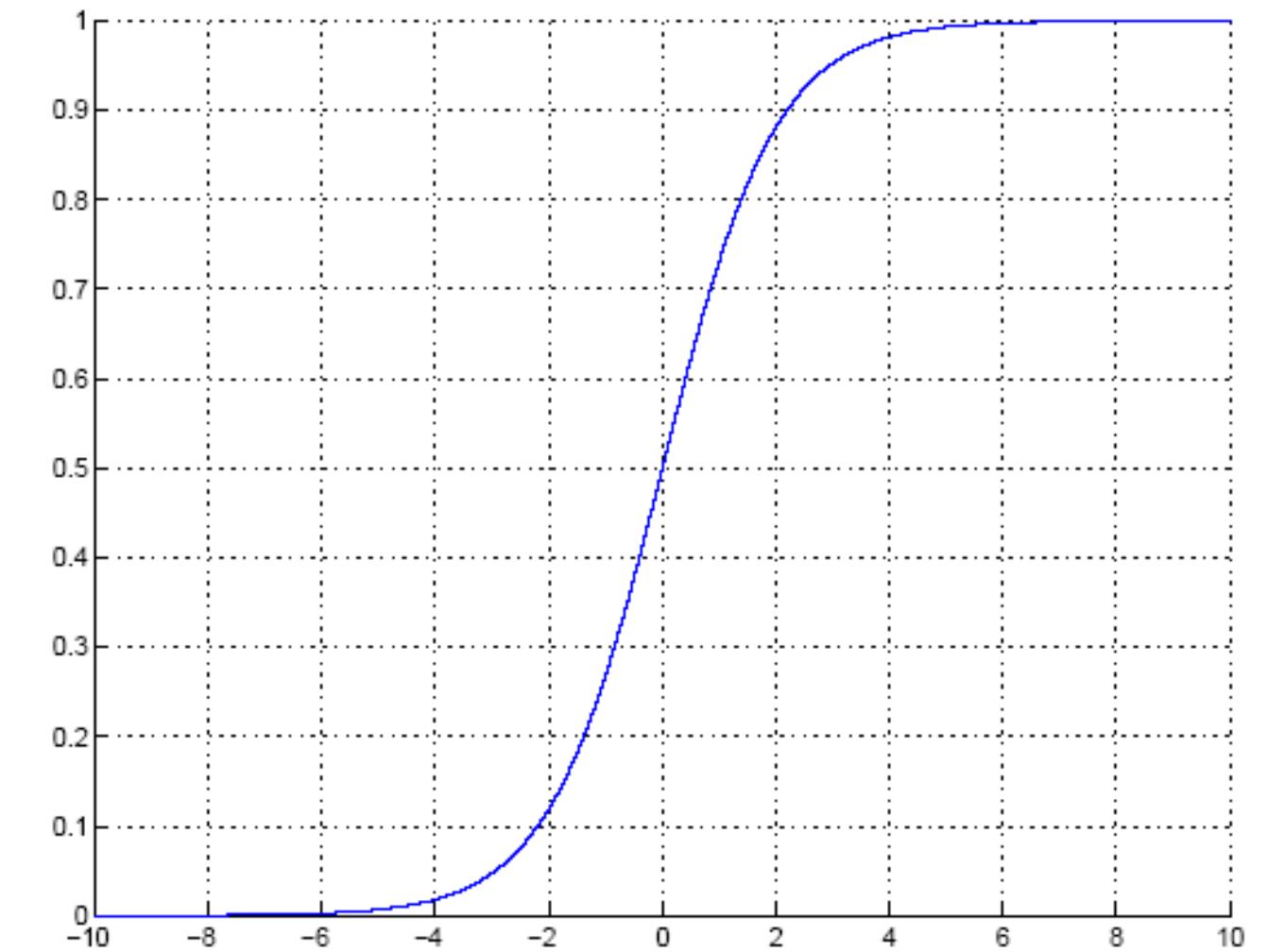
Bernoulli-type conditional distribution

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

Particular choice of form of f :

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

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



Form of posterior distribution

Bernoulli-type conditional distribution

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

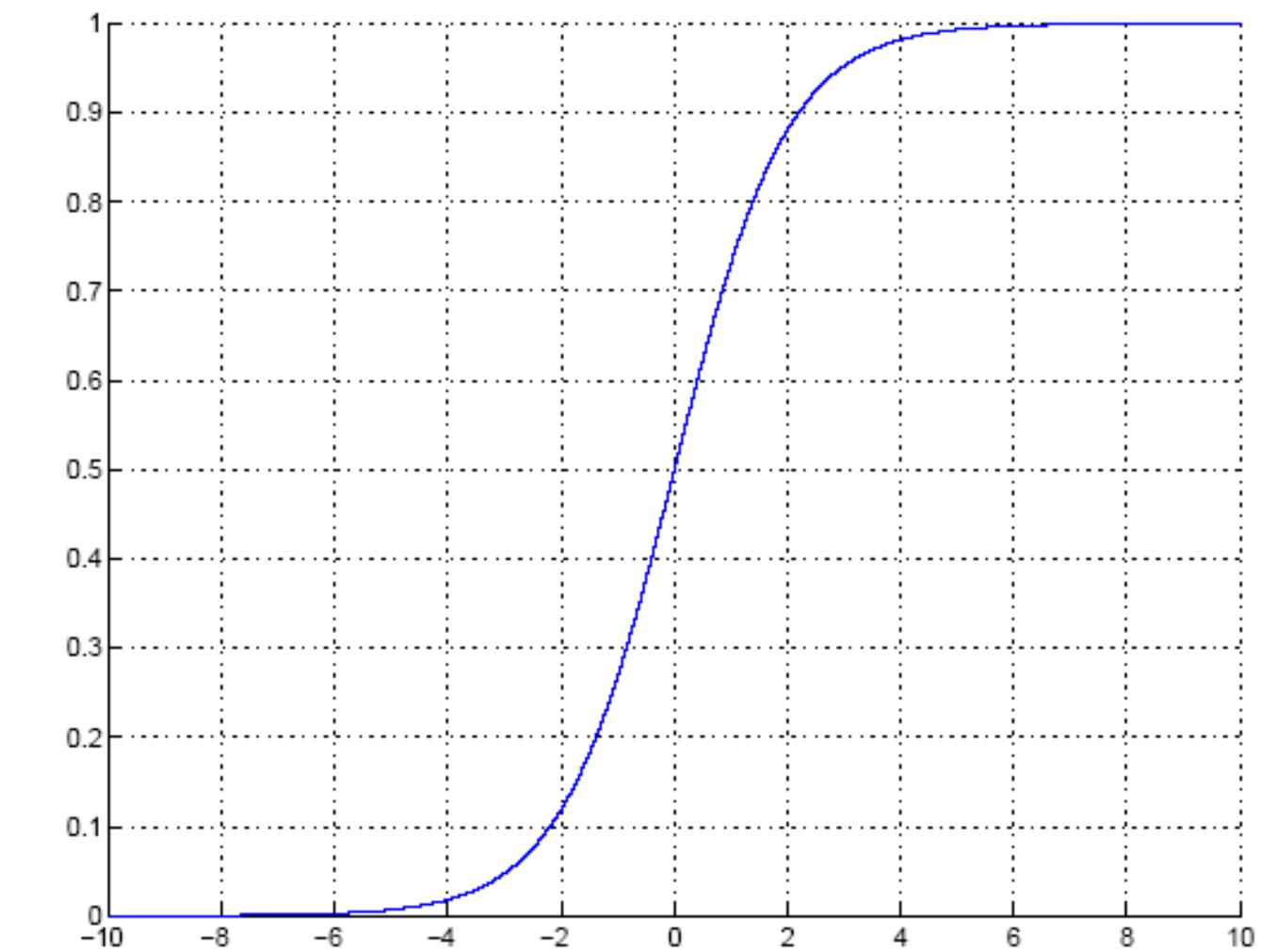
Particular choice of form of f :

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

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

“squashing function”:

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



Form of posterior distribution

Bernoulli-type conditional distribution

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

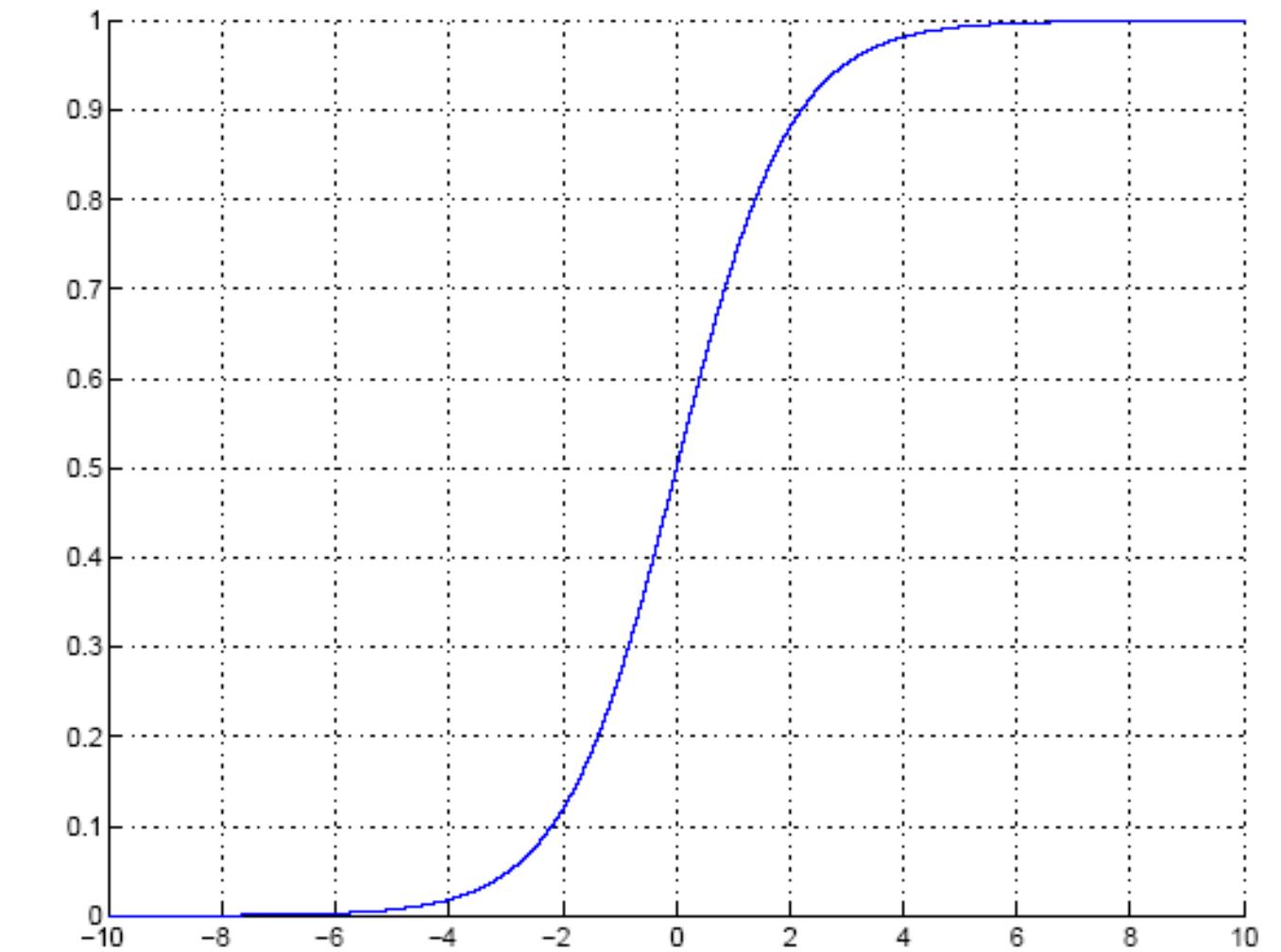
Particular choice of form of f :

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

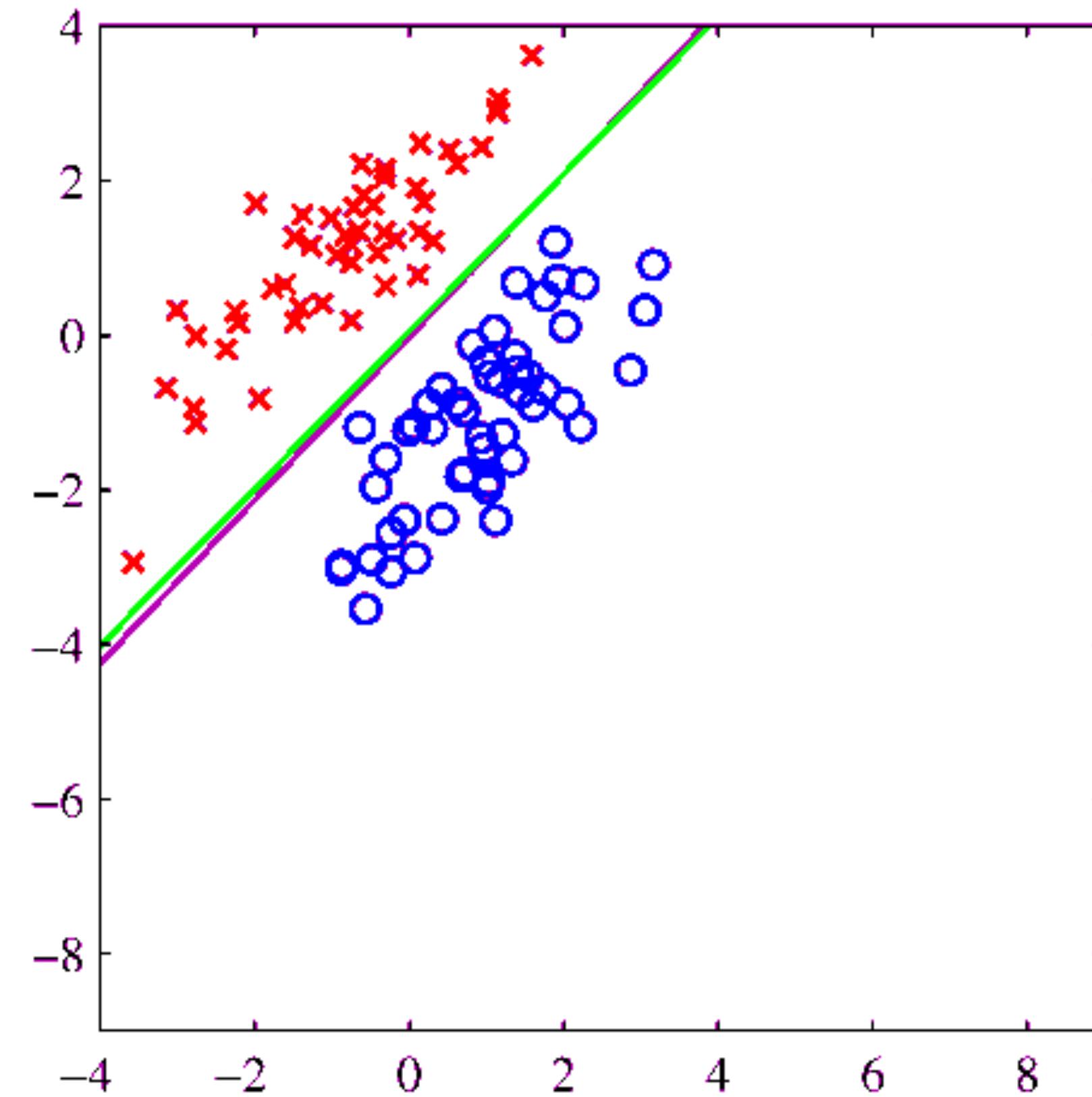
Sigmoidal:
$$g(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$

“squashing function”:

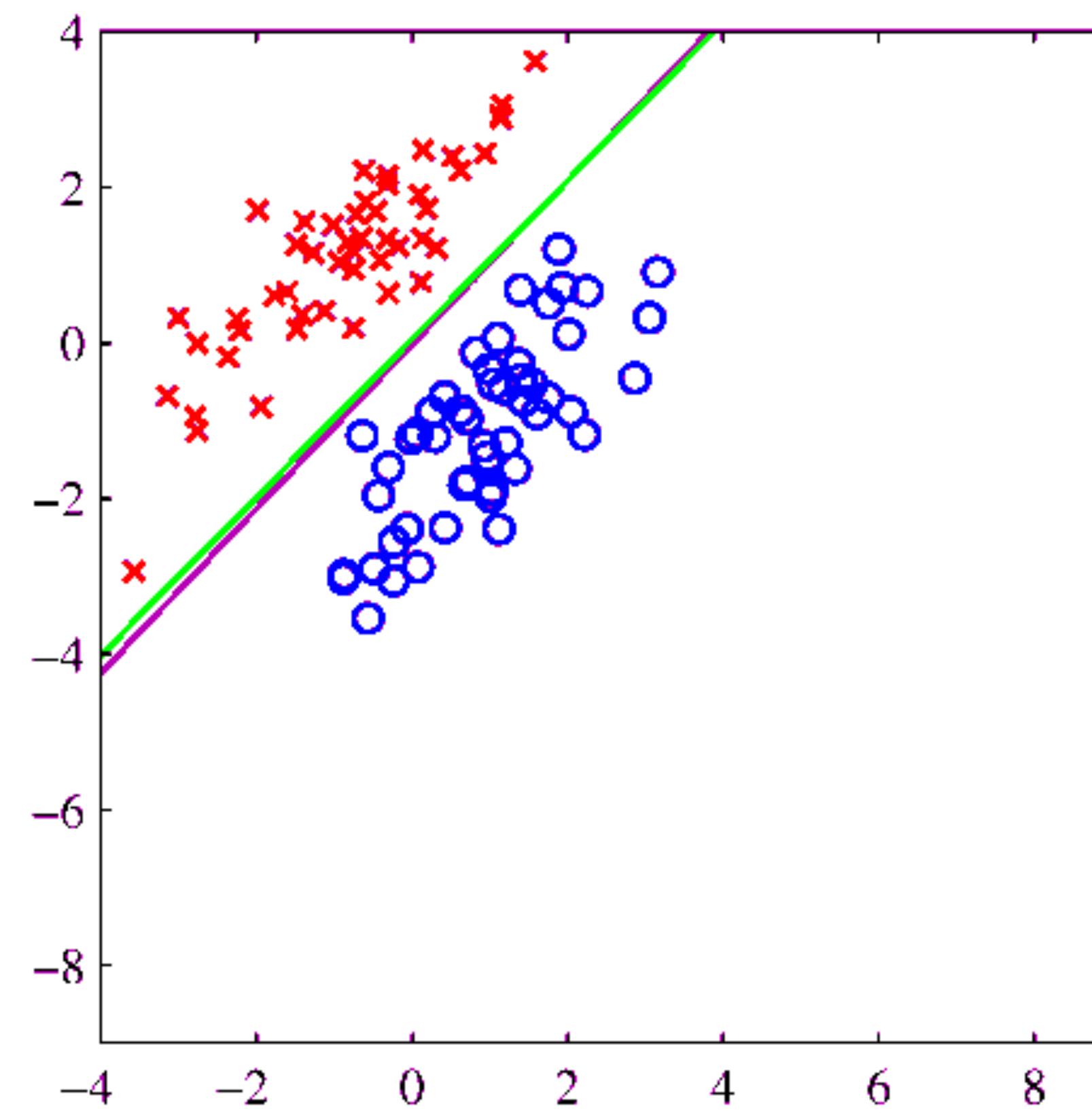
$-\infty \rightarrow 0$
$+\infty \rightarrow 1$



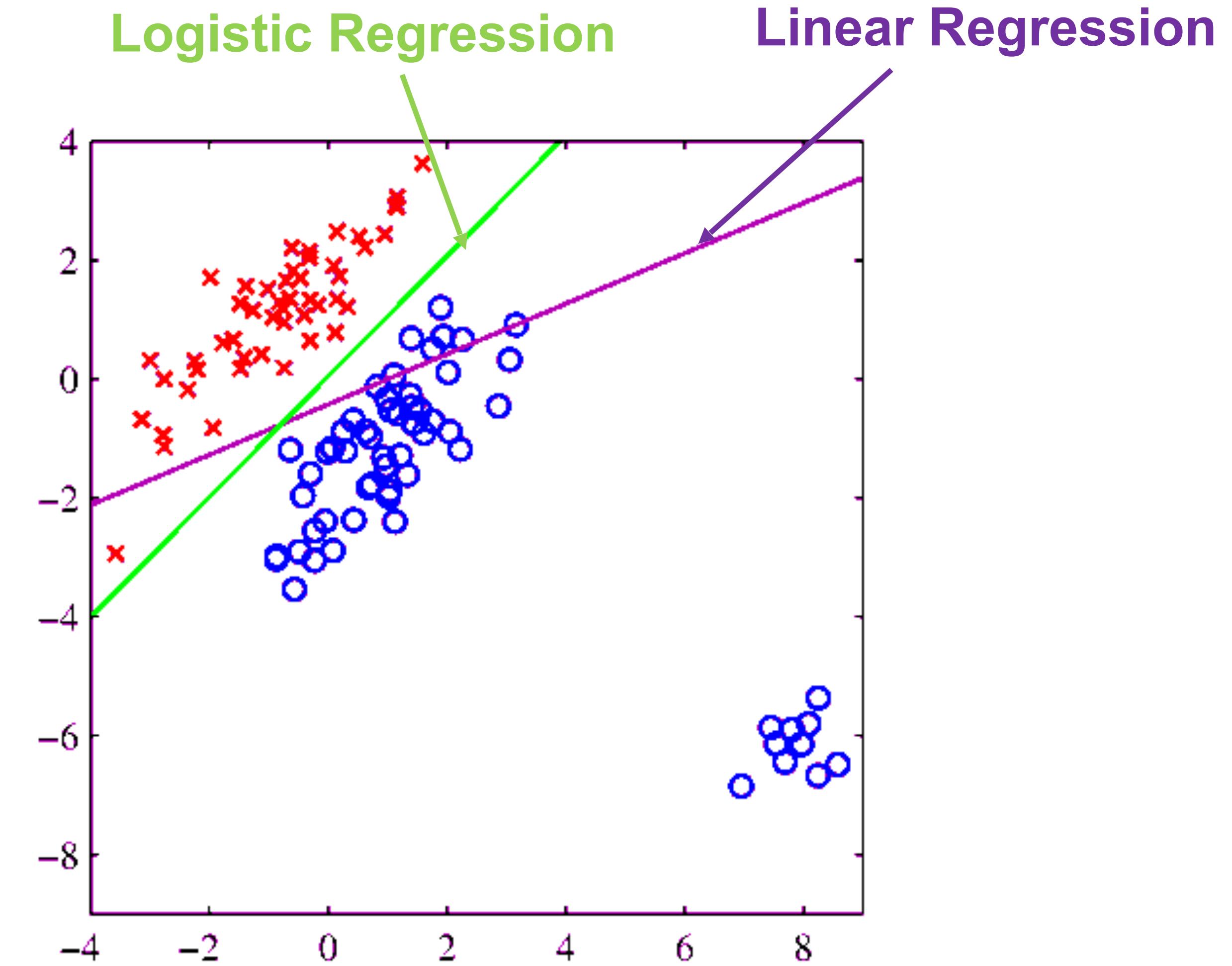
Logistic vs Linear Regression



Logistic vs Linear Regression



Logistic Regression



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:
 $y^i = (0, 0, 1, 0)$

Multiple Classes & Linear Regression

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

4 classes, i-th sample is in 3rd class:
 $y^i = (0, 0, 1, 0)$

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:

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

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:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \vdots \\ \mathbf{y}^N \end{bmatrix} = [\mathbf{y}_1 \mid \dots \mid \mathbf{y}_C] \quad \text{where } \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

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:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \vdots \\ \mathbf{y}^N \end{bmatrix} = [\mathbf{y}_1 \mid \dots \mid \mathbf{y}_C] \quad \text{where } \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

$$\mathbf{W} = [\mathbf{w}_1 \mid \dots \mid \mathbf{w}_C]$$

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:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \vdots \\ \mathbf{y}^N \end{bmatrix} = [\mathbf{y}_1 \mid \dots \mid \mathbf{y}_C] \quad \text{where } \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

$$\mathbf{W} = [\mathbf{w}_1 \mid \dots \mid \mathbf{w}_C]$$

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:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \vdots \\ \mathbf{y}^N \end{bmatrix} = [\mathbf{y}_1 \mid \dots \mid \mathbf{y}_C] \quad \text{where } \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

$$\mathbf{W} = [\mathbf{w}_1 \mid \dots \mid \mathbf{w}_C]$$

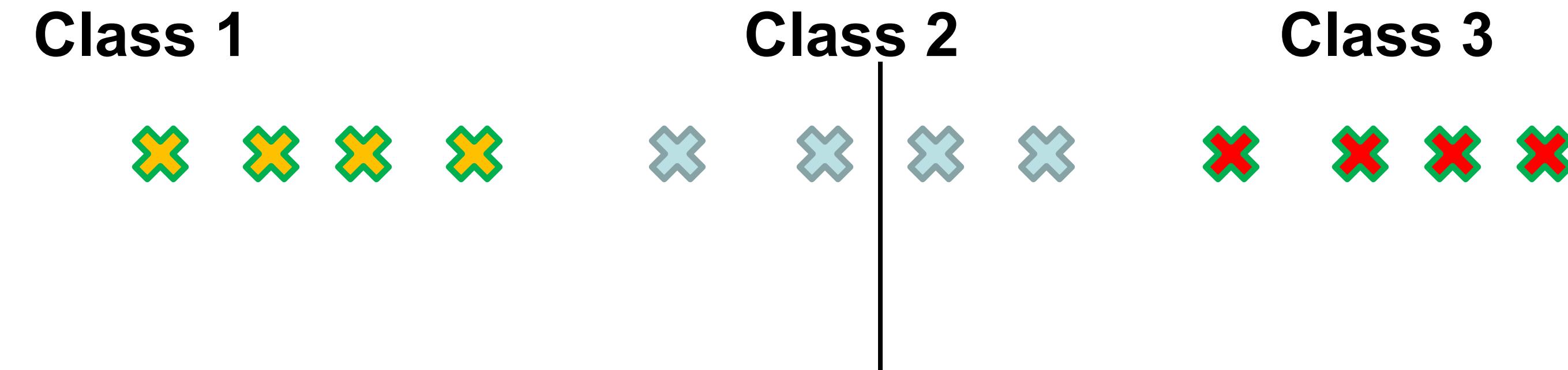
Loss function:

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

Least squares fit (decouples per class):

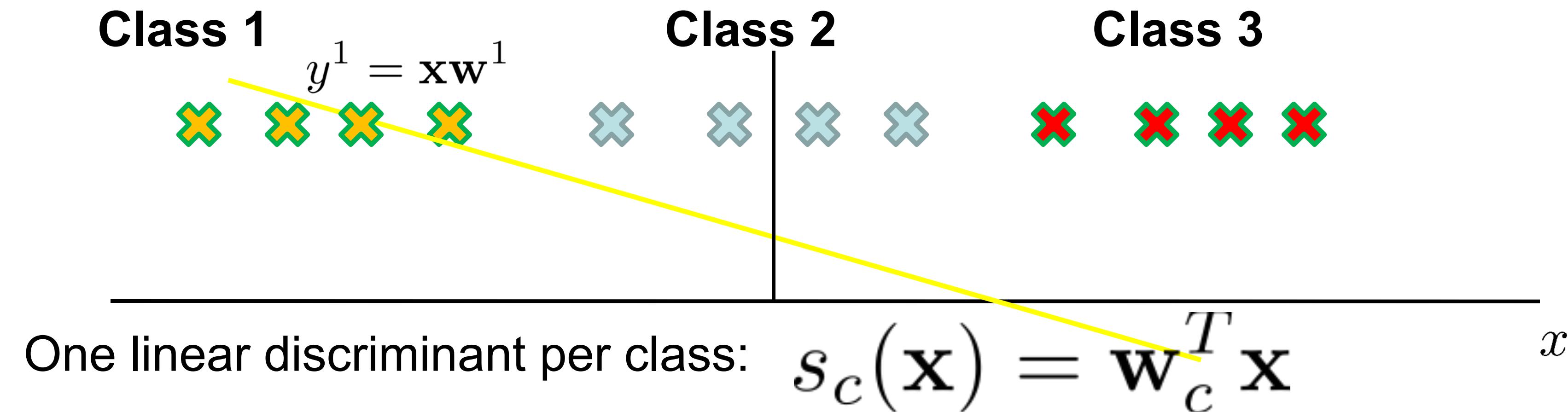
$$\mathbf{w}_c^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}_c$$

Linear Regression Masking Problem

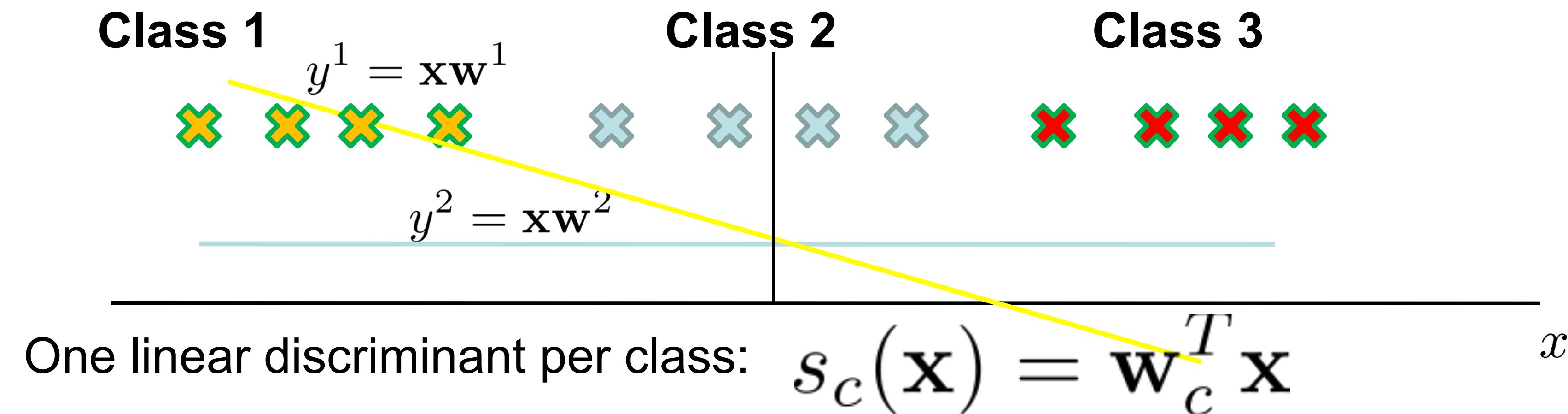


One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

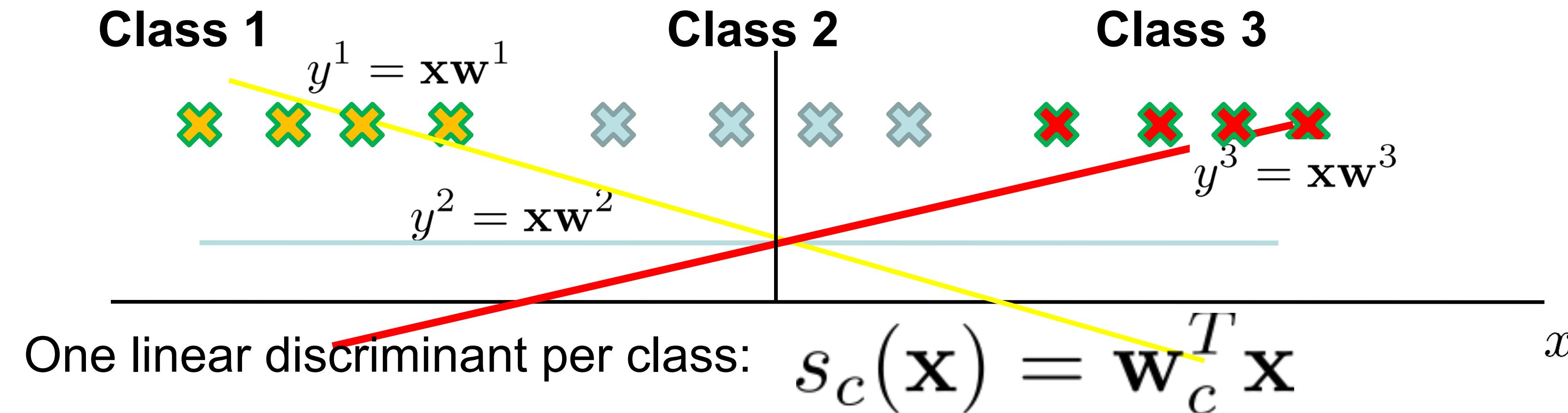
Linear Regression Masking Problem



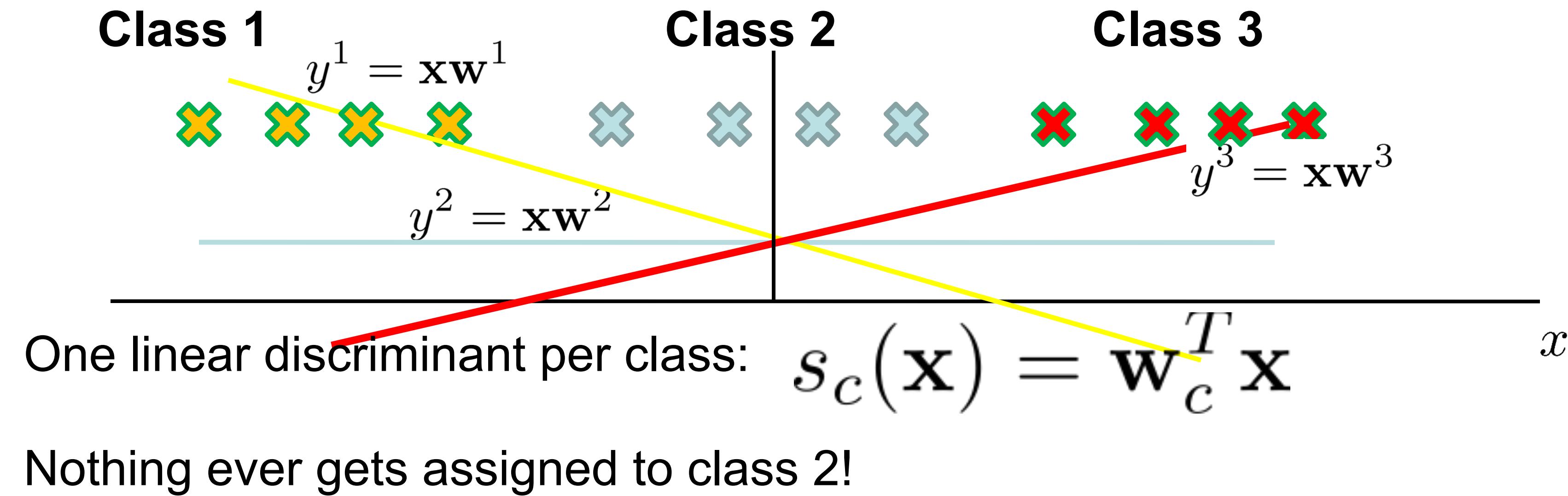
Linear Regression Masking Problem



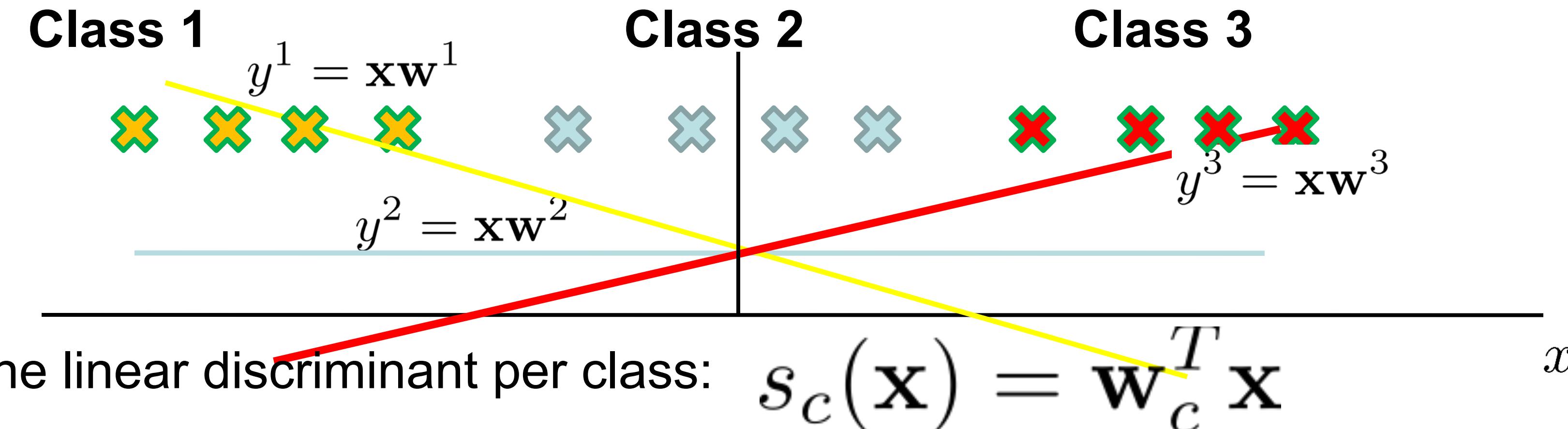
Linear Regression Masking Problem



Linear Regression Masking Problem

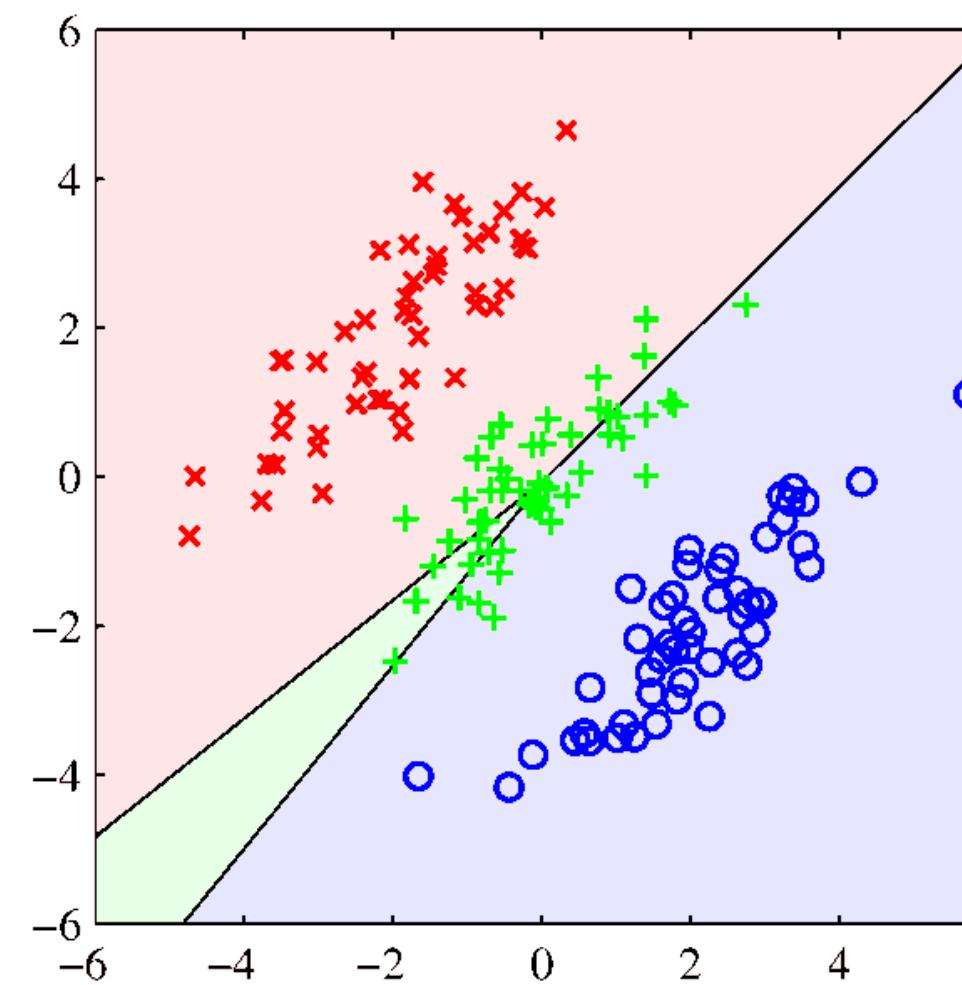


Linear Regression Masking Problem



Nothing ever gets assigned to class 2!

2D version:



Multiple classes & Logistic regression

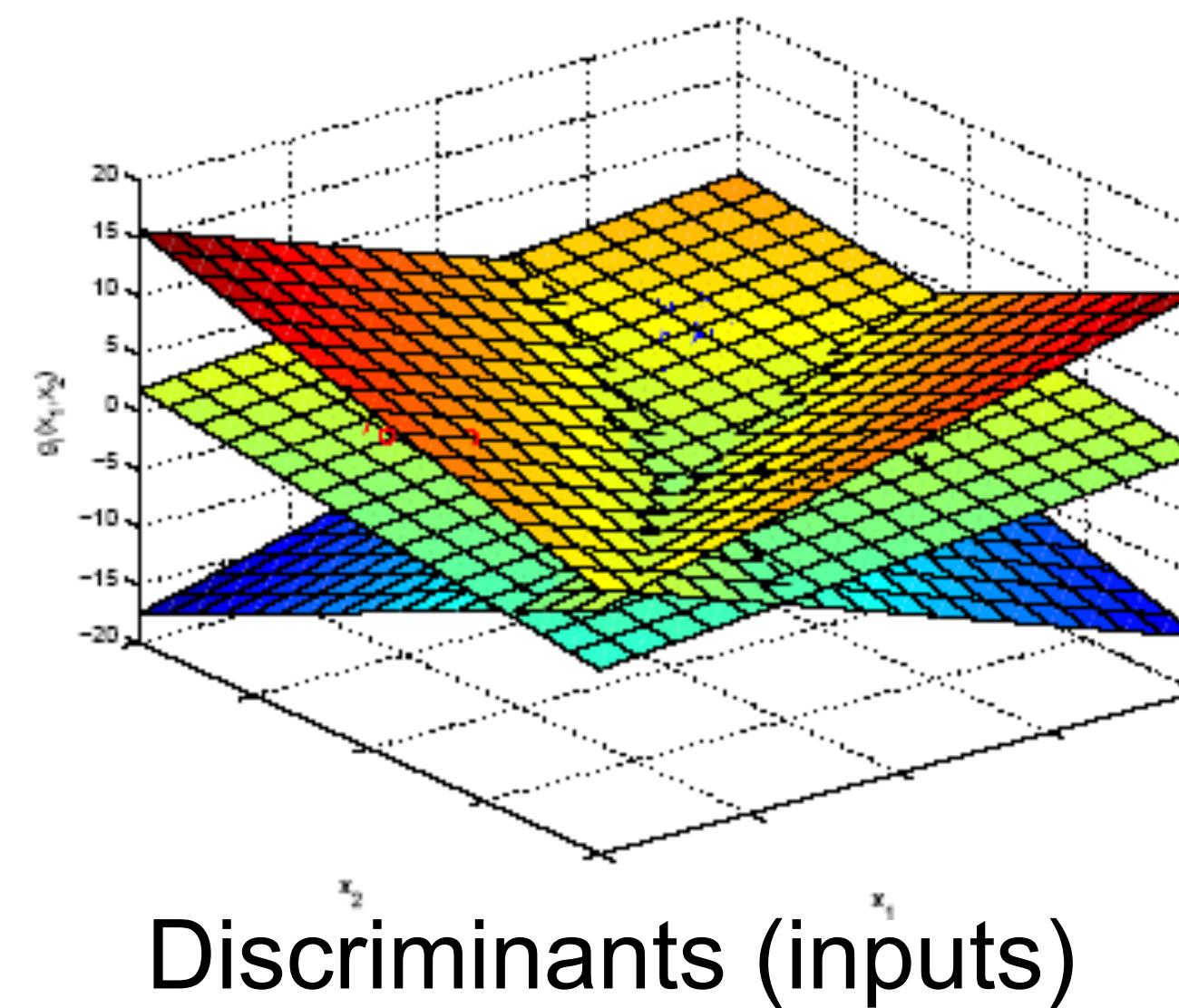
Soft maximum (softmax) of competing classes:

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

Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

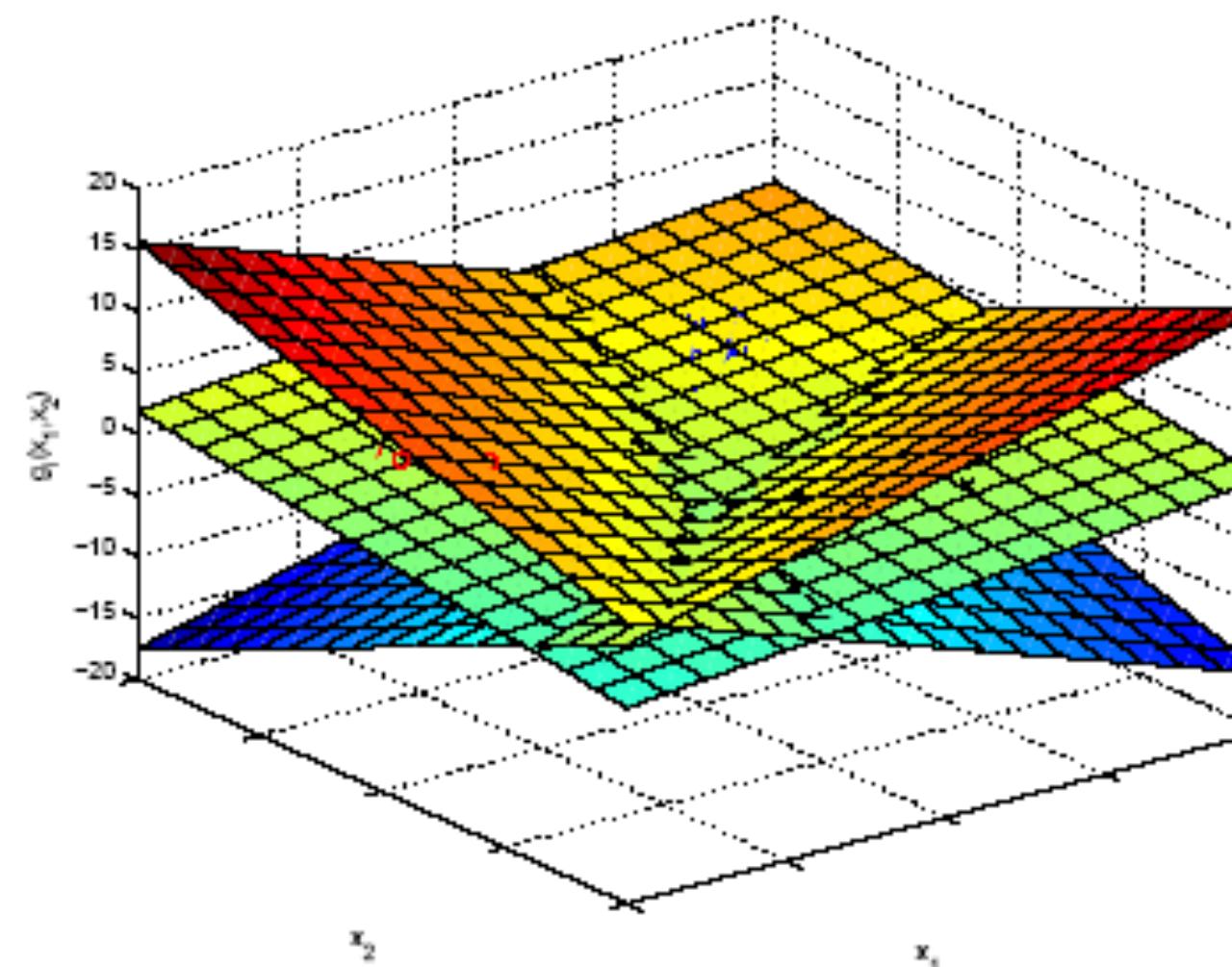
$$P(y = c|\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^C \exp(\mathbf{w}_{c'}^T \mathbf{x})} \doteq g_c(\mathbf{x}, \mathbf{W})$$



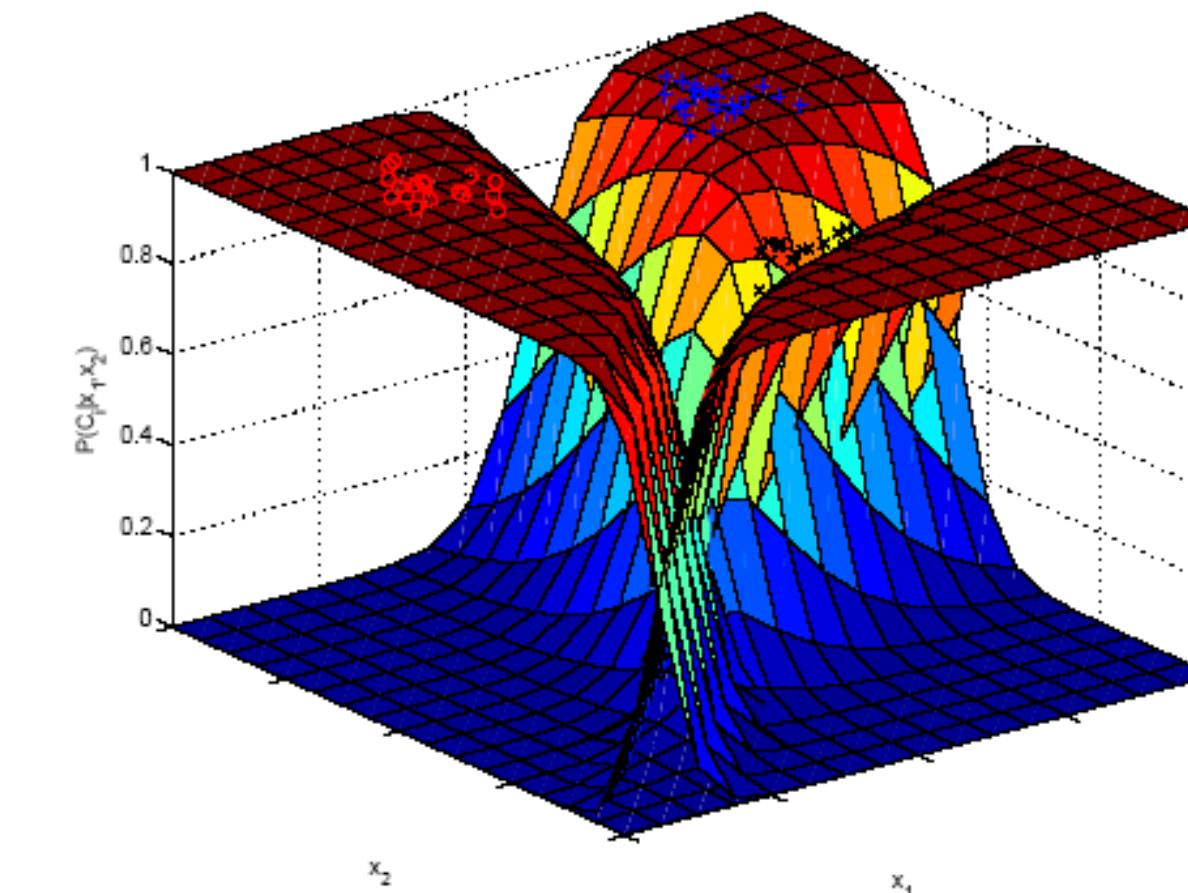
Multiple classes & Logistic regression

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)

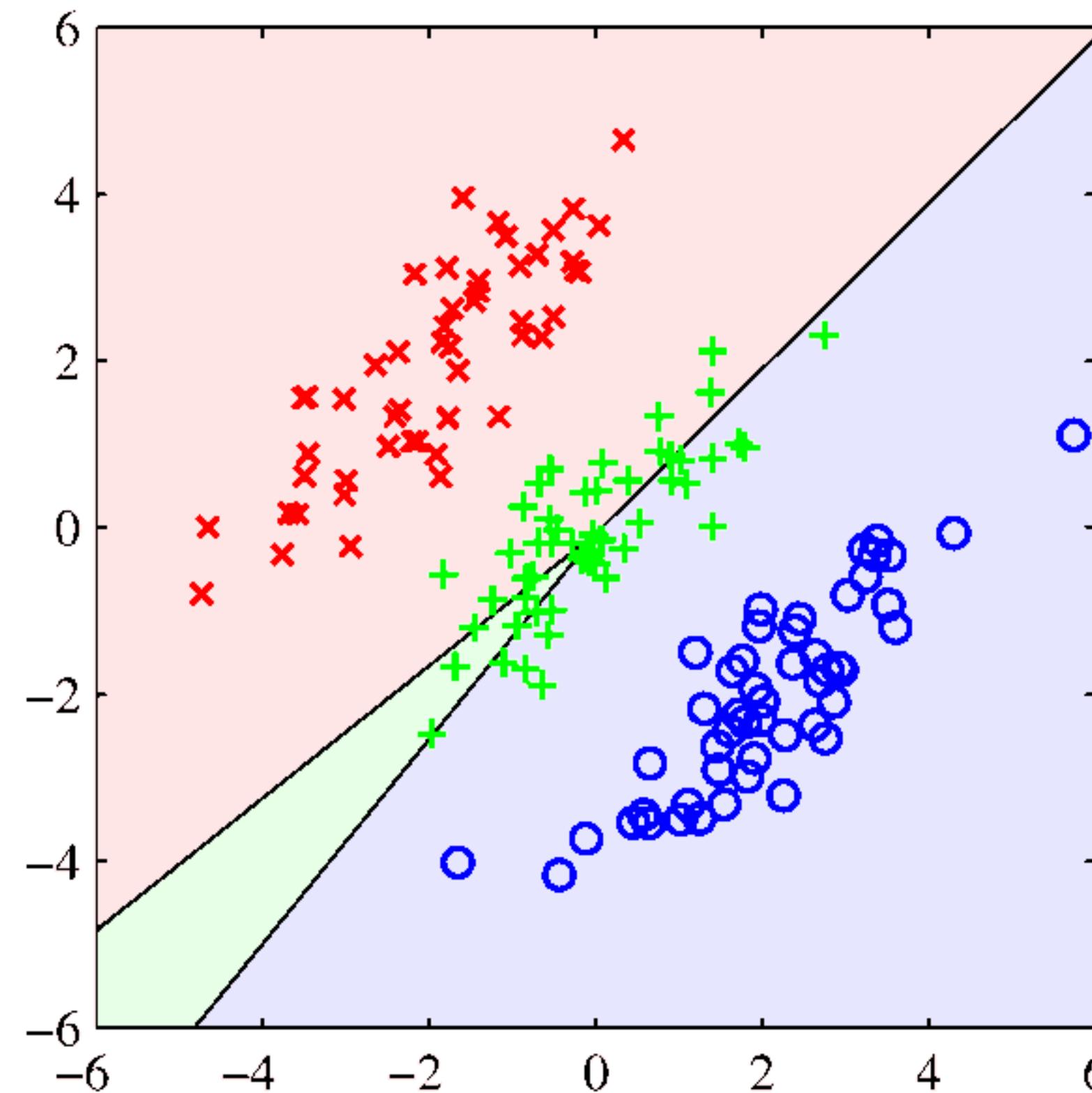


Softmax (outputs)

Logistic vs Linear Regression, $n>2$ classes

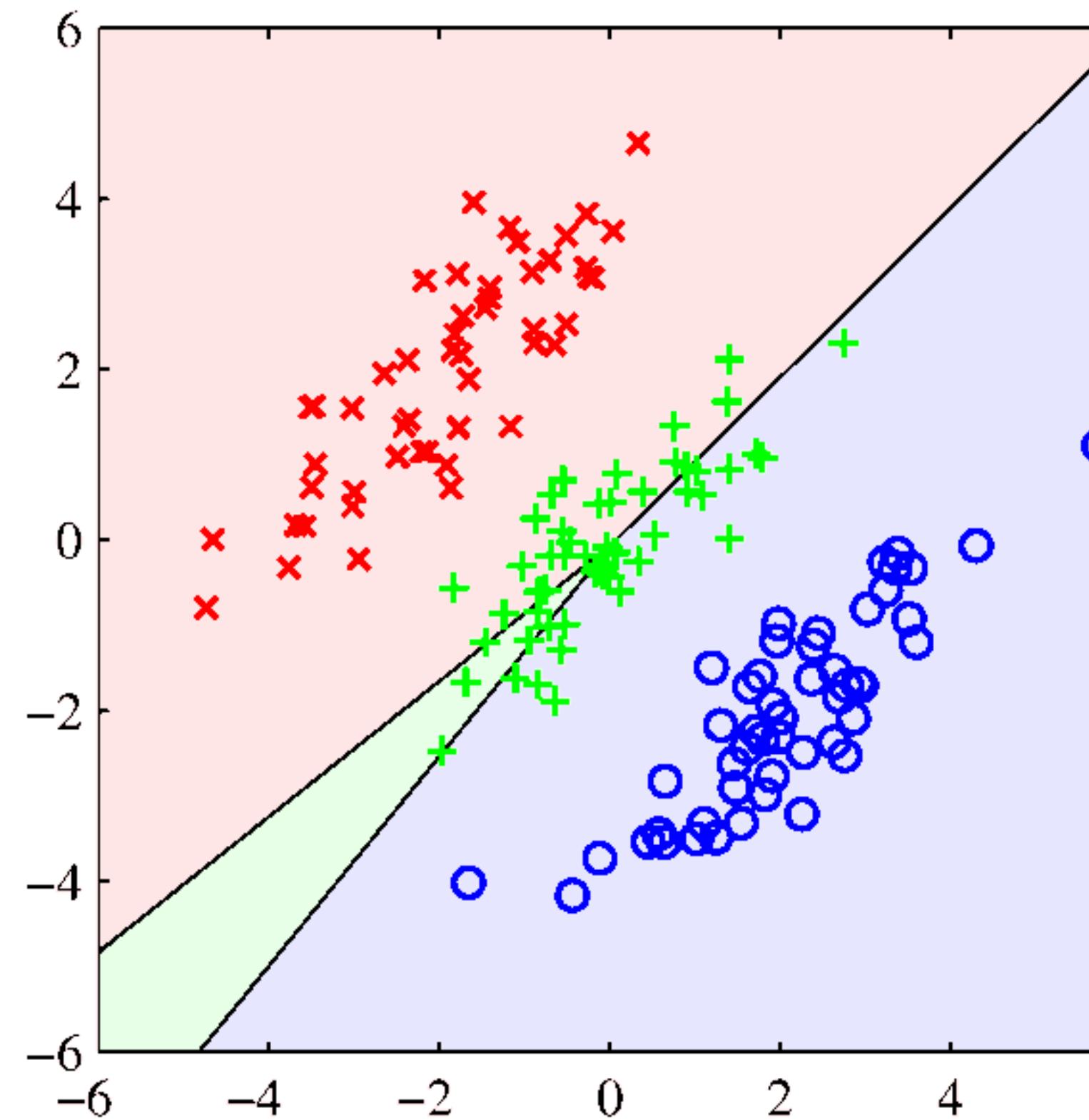
Logistic vs Linear Regression, n>2 classes

Linear regression

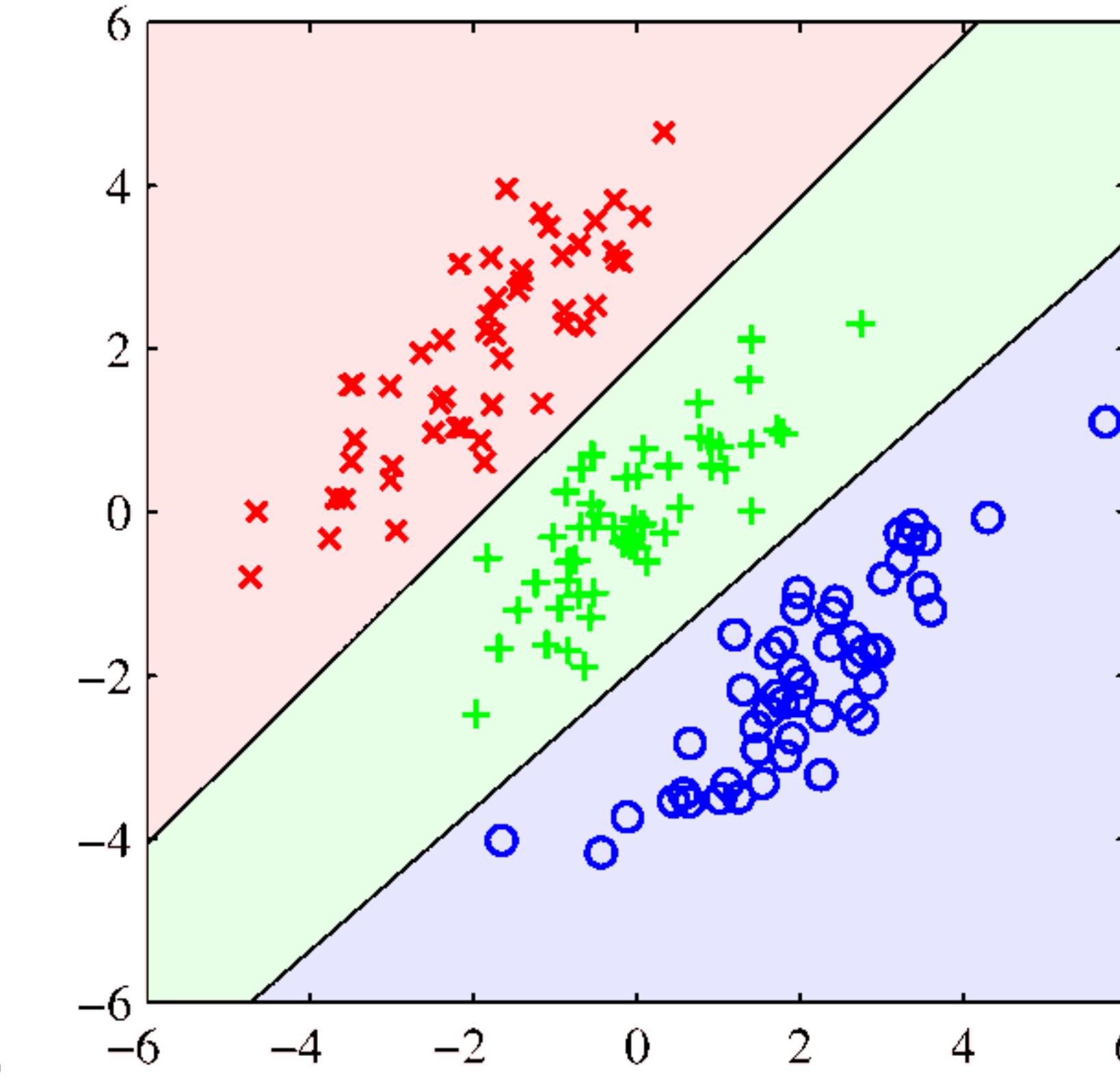


Logistic vs Linear Regression, n>2 classes

Linear regression

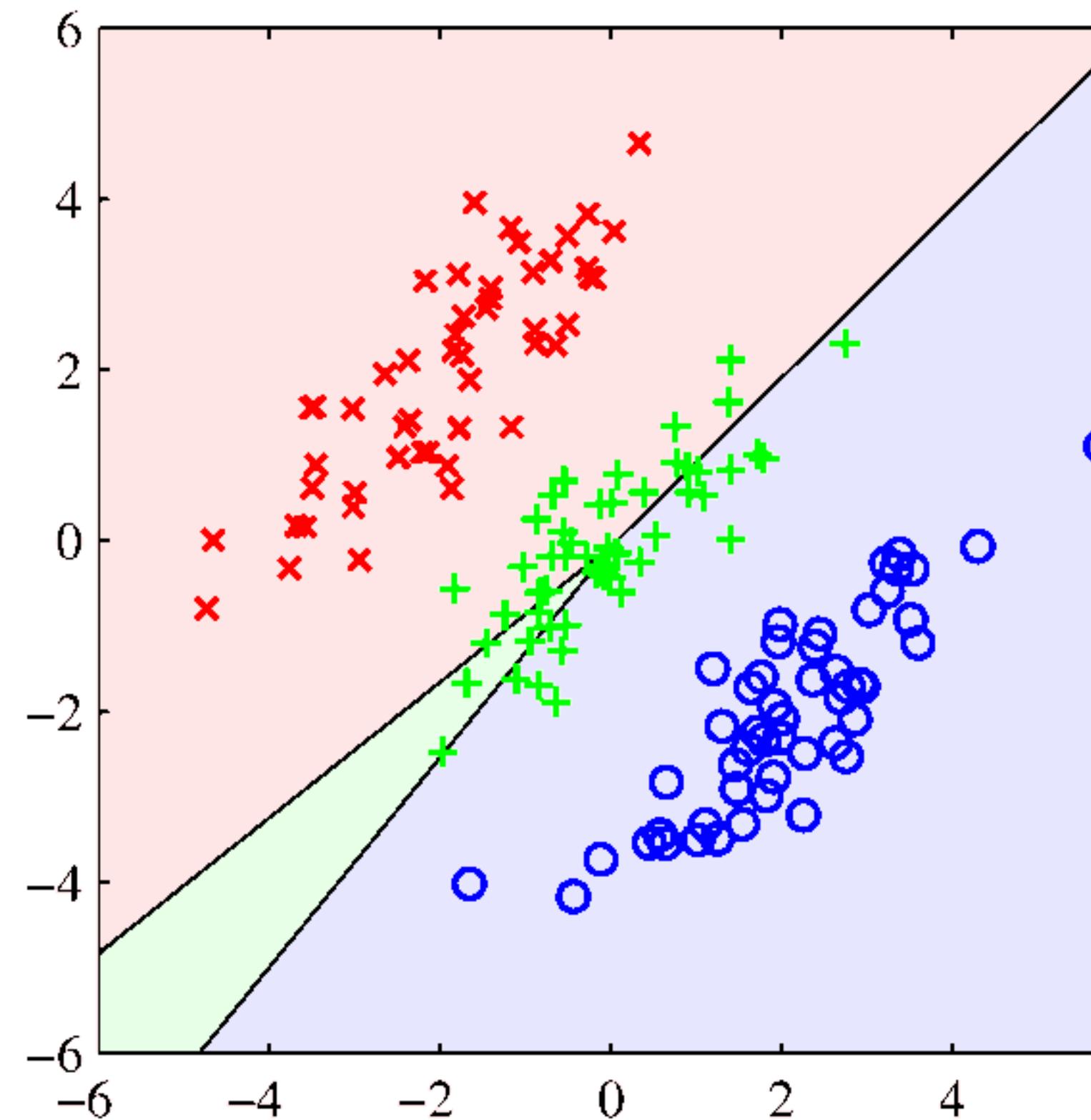


Logistic regression

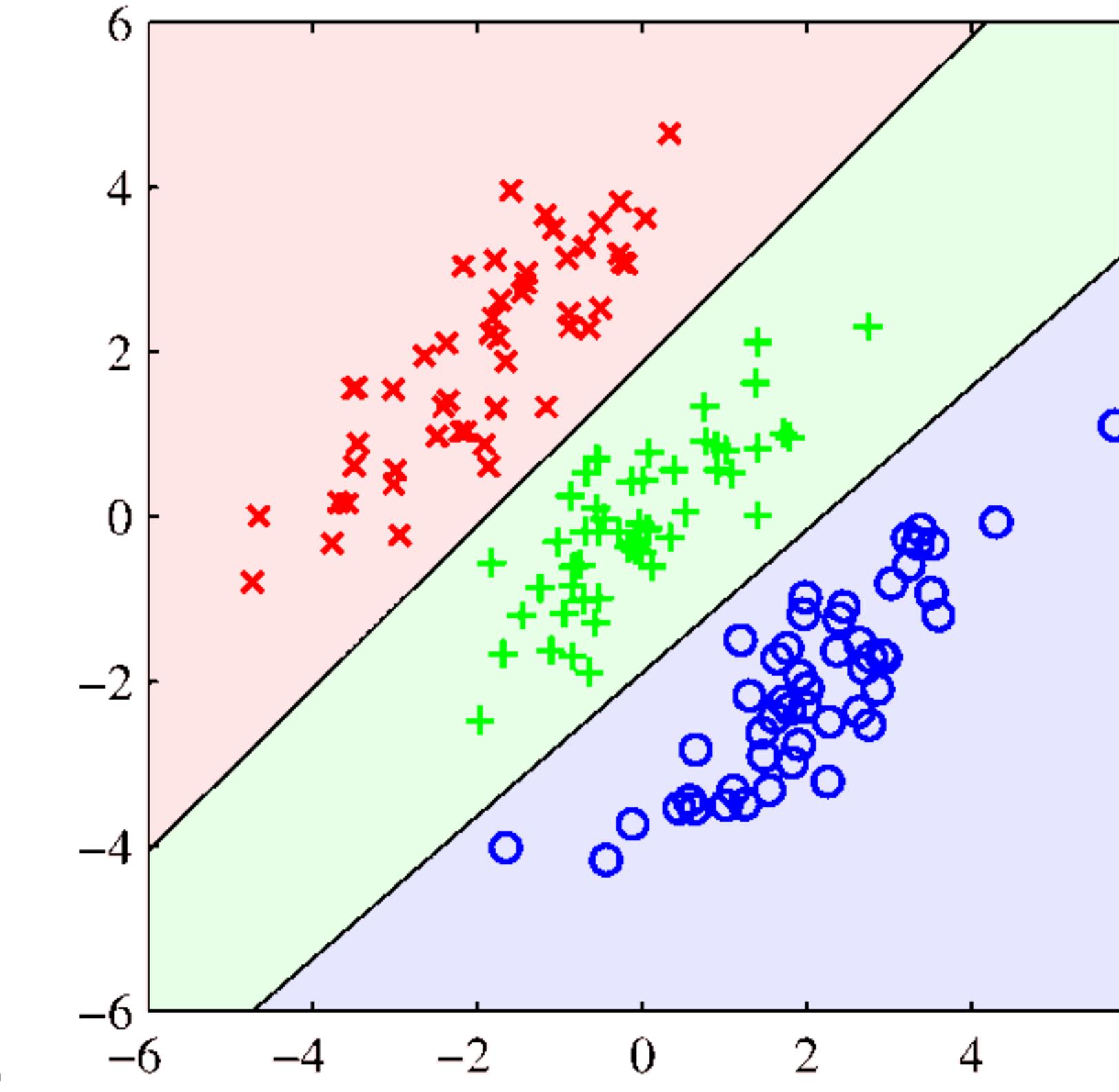


Logistic vs Linear Regression, $n > 2$ classes

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)

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

LS Solution (in vector form)

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

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)

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

Condition for minimum:

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

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

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

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

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

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

using

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

$$\begin{aligned} &= - \sum_{i=1}^N \left[y^i \frac{1}{g(\mathbf{w}^T \mathbf{x}^i)} - (1 - y^i) \frac{1}{1 - g(\mathbf{w}^T \mathbf{x}^i)} \right] g(\mathbf{w}^T \mathbf{x}^i)(1 - g(\mathbf{w}^T \mathbf{x}^i)) \frac{\partial \mathbf{w}^T \mathbf{x}^i}{\partial w_k} \\ &= - \sum_{i \neq 1} \left[y^i (1 - g(\mathbf{w}^T \mathbf{x}^i)) - (1 - y^i) g(\mathbf{w}^T \mathbf{x}^i) \right] x_k^i \\ &= - \sum_{i=1}^N [y^i - g(\mathbf{w}^T \mathbf{x}^i)] \mathbf{x}_k^i \end{aligned}$$

Gradient of Cross-entropy Loss

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

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

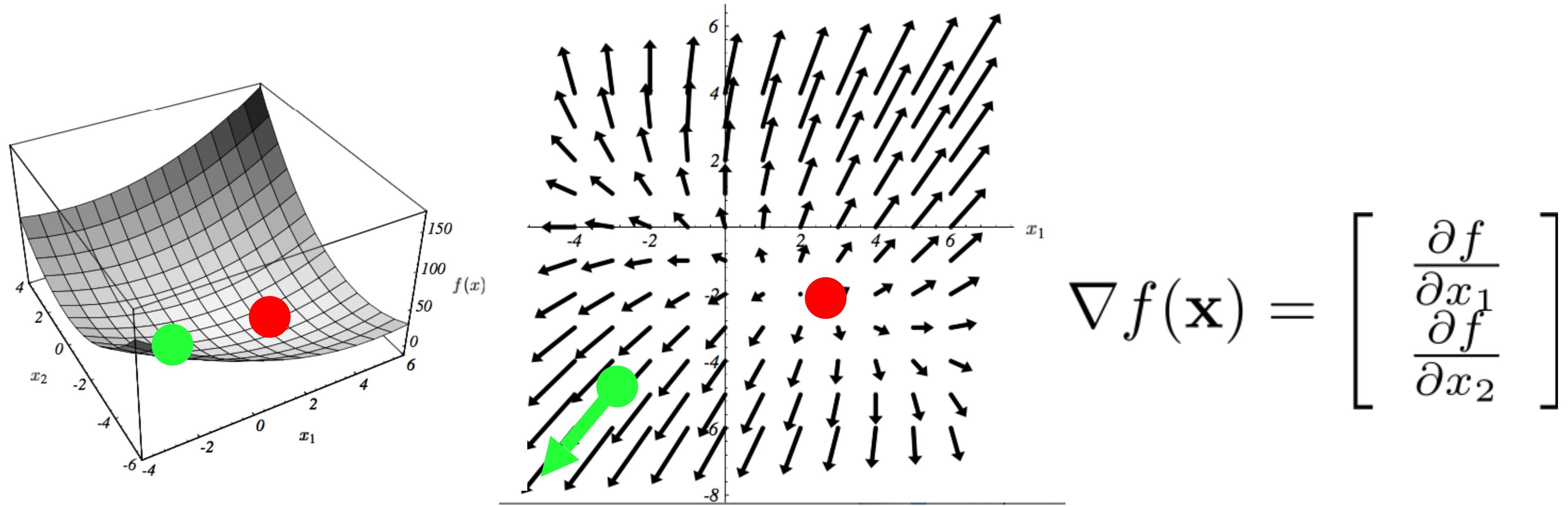
using

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

$$\begin{aligned} &= - \sum_{i=1}^N \left[y^i \frac{1}{g(\mathbf{w}^T \mathbf{x}^i)} - (1 - y^i) \frac{1}{1 - g(\mathbf{w}^T \mathbf{x}^i)} \right] g(\mathbf{w}^T \mathbf{x}^i)(1 - g(\mathbf{w}^T \mathbf{x}^i)) \frac{\partial \mathbf{w}^T \mathbf{x}^i}{\partial w_k} \\ &= - \sum_{i \neq 1} [y^i(1 - g(\mathbf{w}^T \mathbf{x}^i)) - (1 - y^i)g(\mathbf{w}^T \mathbf{x}^i)] x_k^i \\ &= - \sum_{i=1} [y^i - g(\mathbf{w}^T \mathbf{x}^i)] \mathbf{x}_k^i \end{aligned}$$

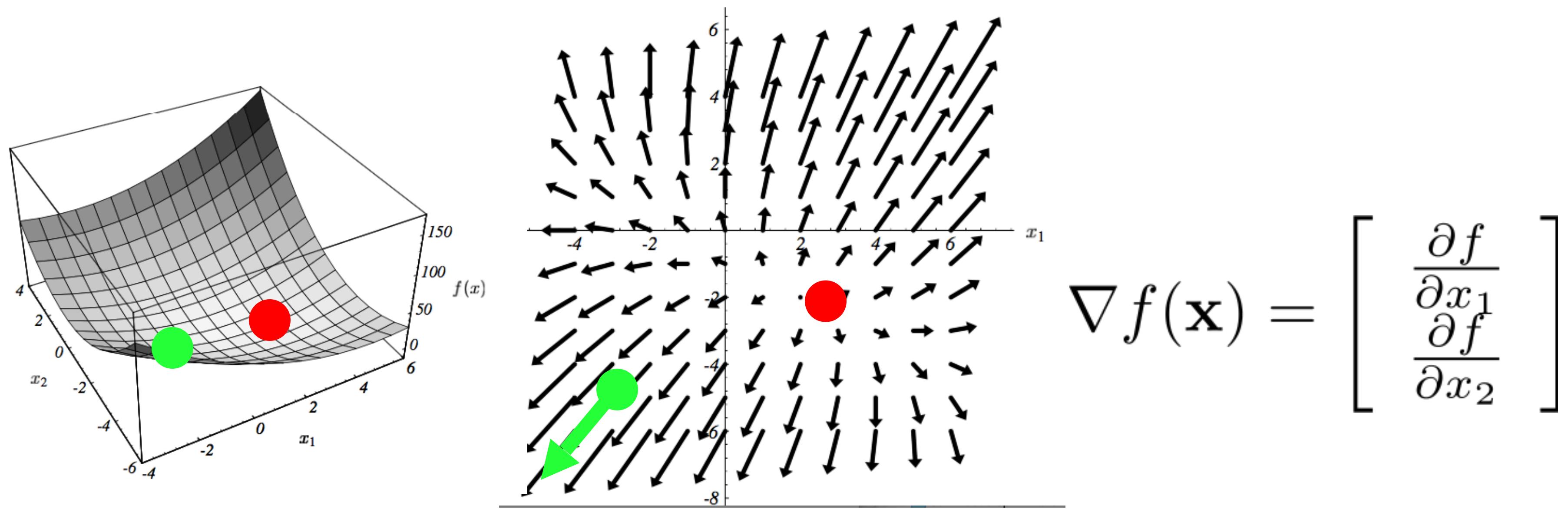
$\nabla L(\mathbf{w}^*) = \mathbf{0}$
nonlinear system of equations!!

Gradient Descent Minimization



Fact: gradient at any point gives direction of fastest increase

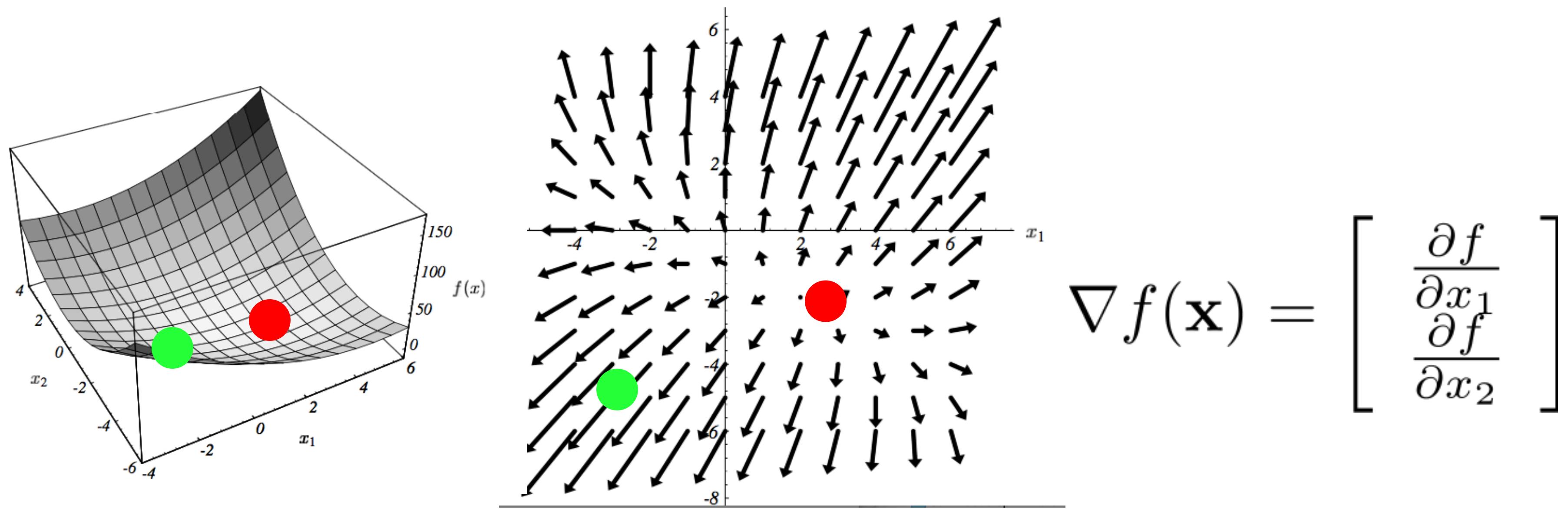
Gradient Descent Minimization



Fact: gradient at any point gives direction of fastest increase

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

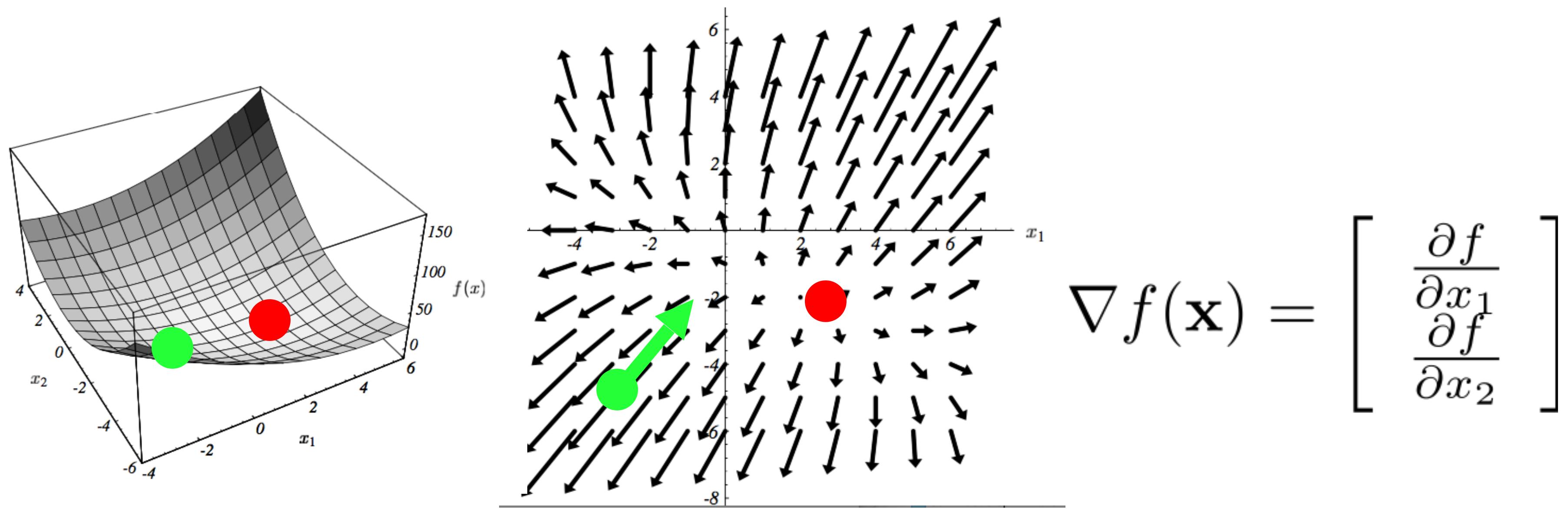
Gradient Descent Minimization



Fact: gradient at any point gives direction of fastest increase

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

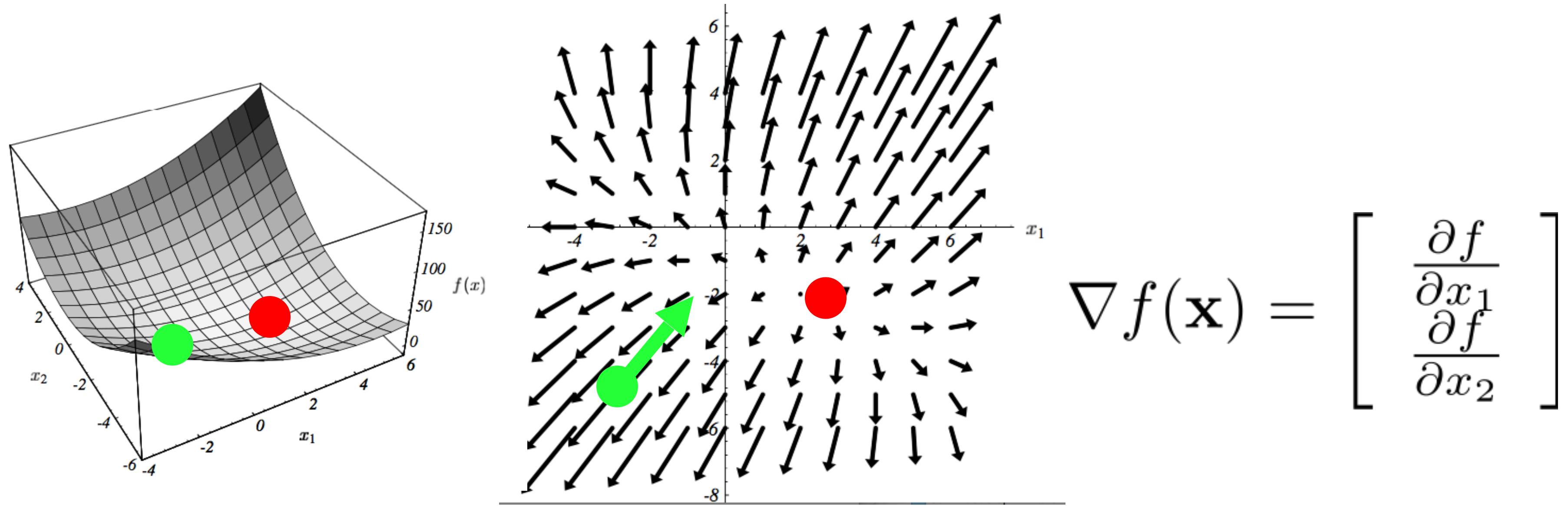
Gradient Descent Minimization



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



Fact: gradient at any point gives direction of fastest increase

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

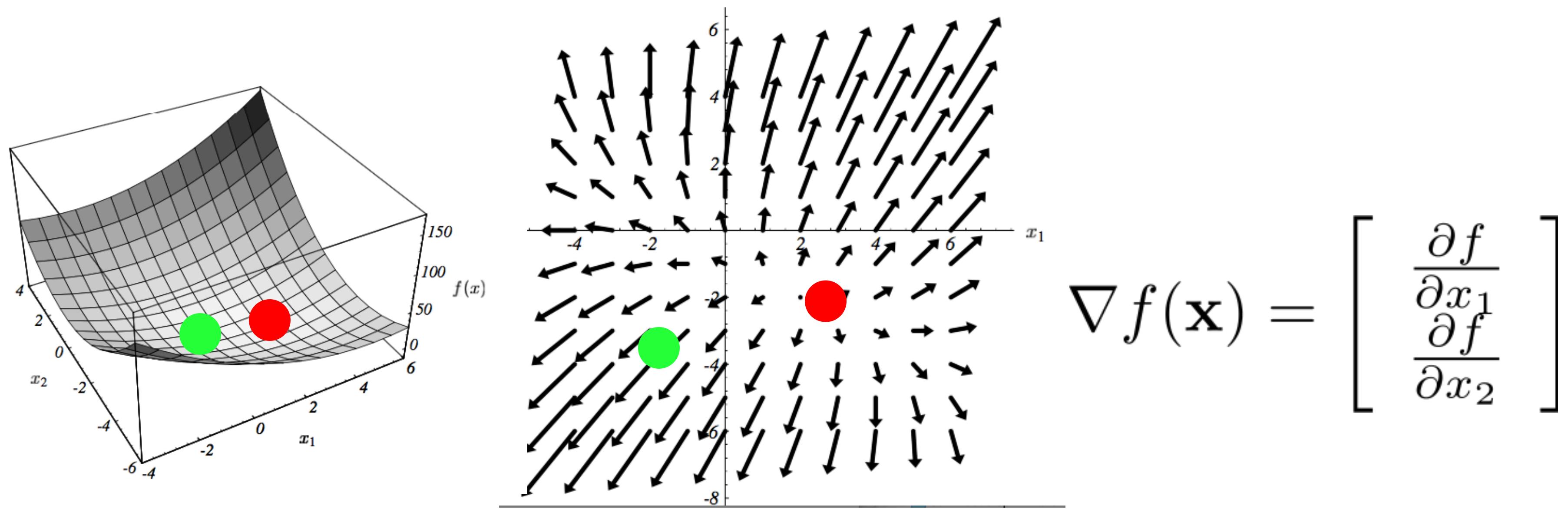
Initialize:

$$\mathbf{x}_0$$

Update:

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

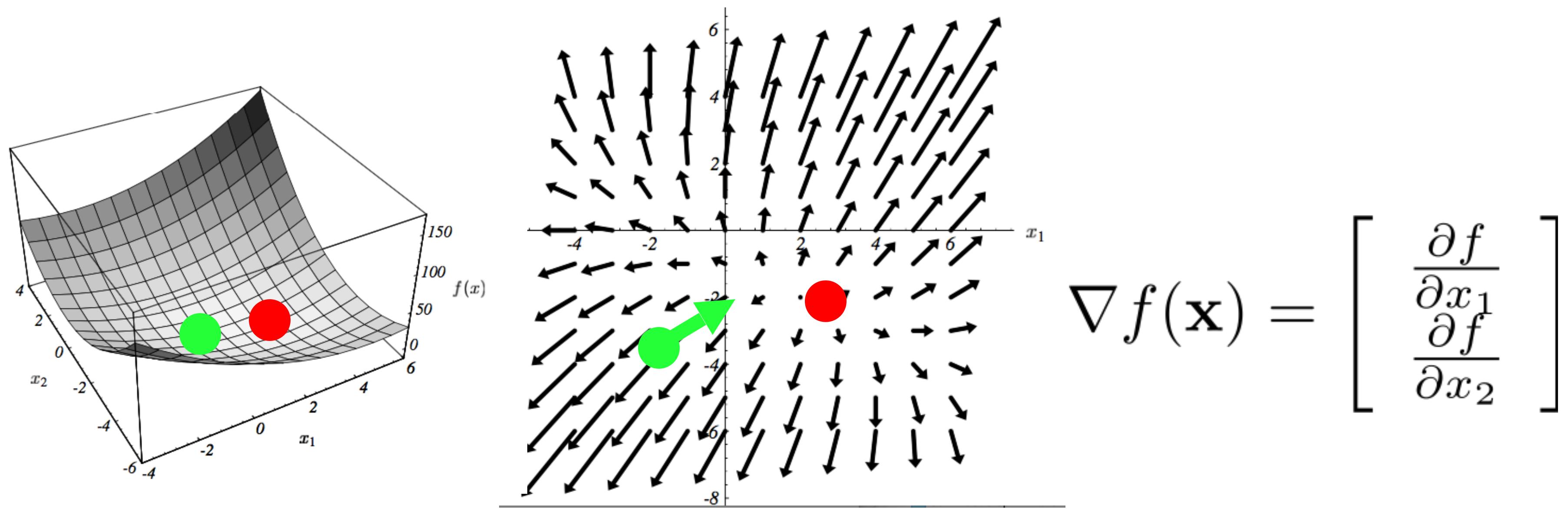
Gradient Descent Minimization



Update:

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

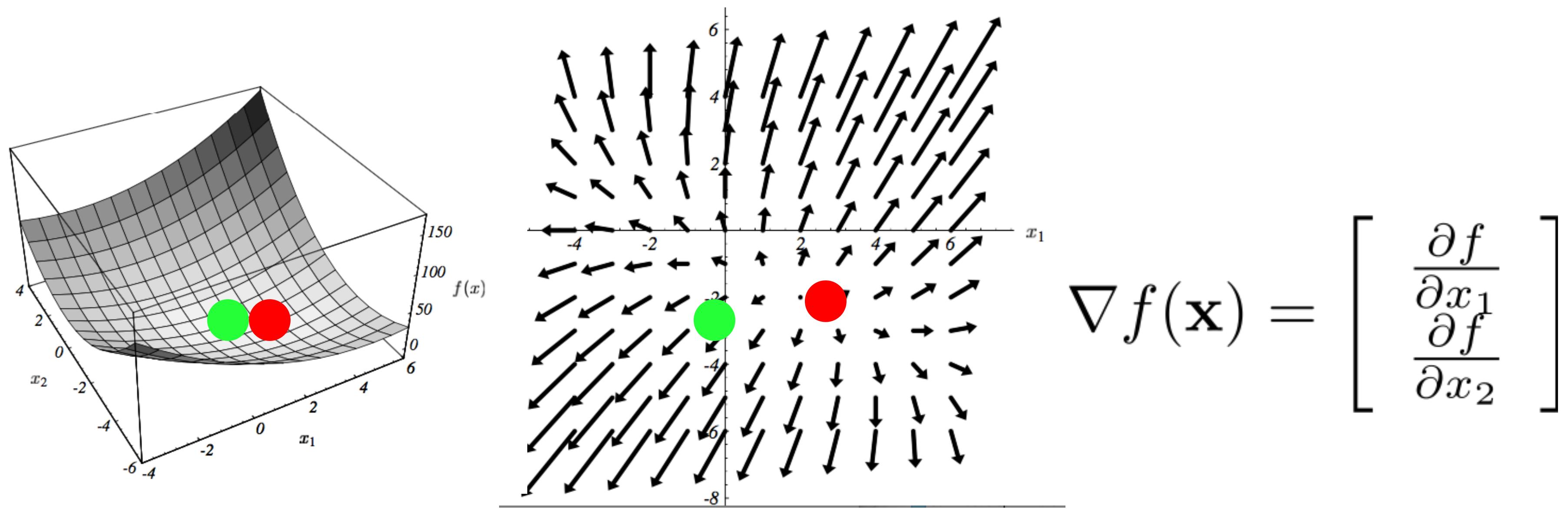
Gradient Descent Minimization



Update:

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

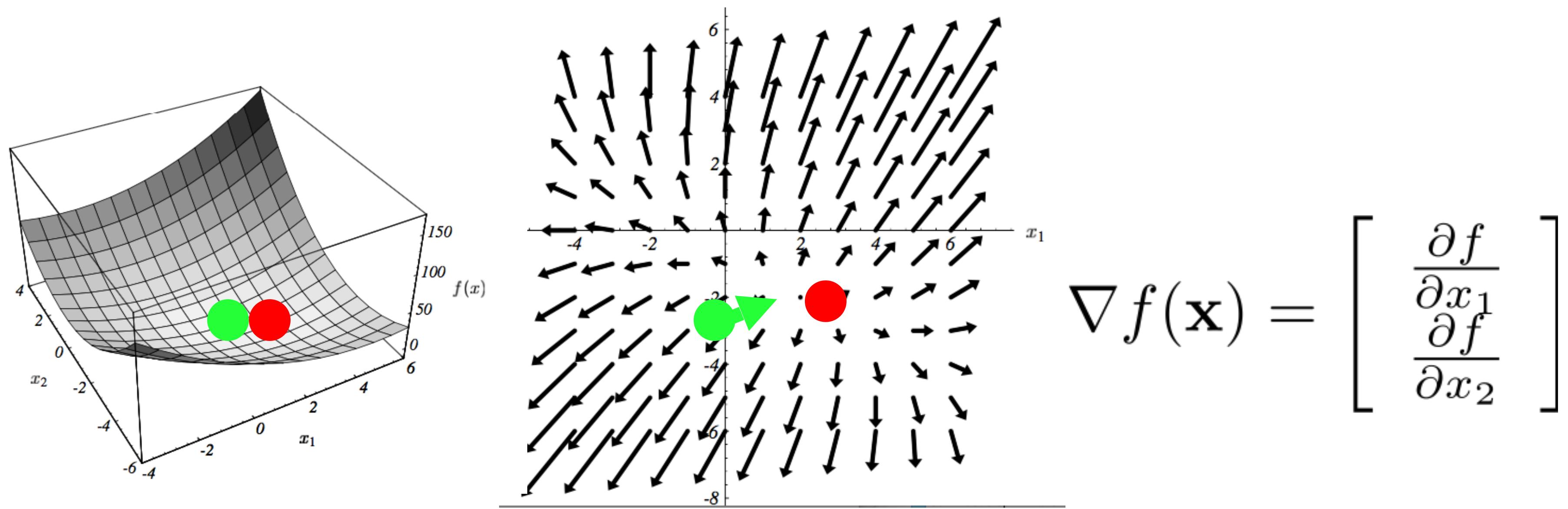
Gradient Descent Minimization



Update:

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

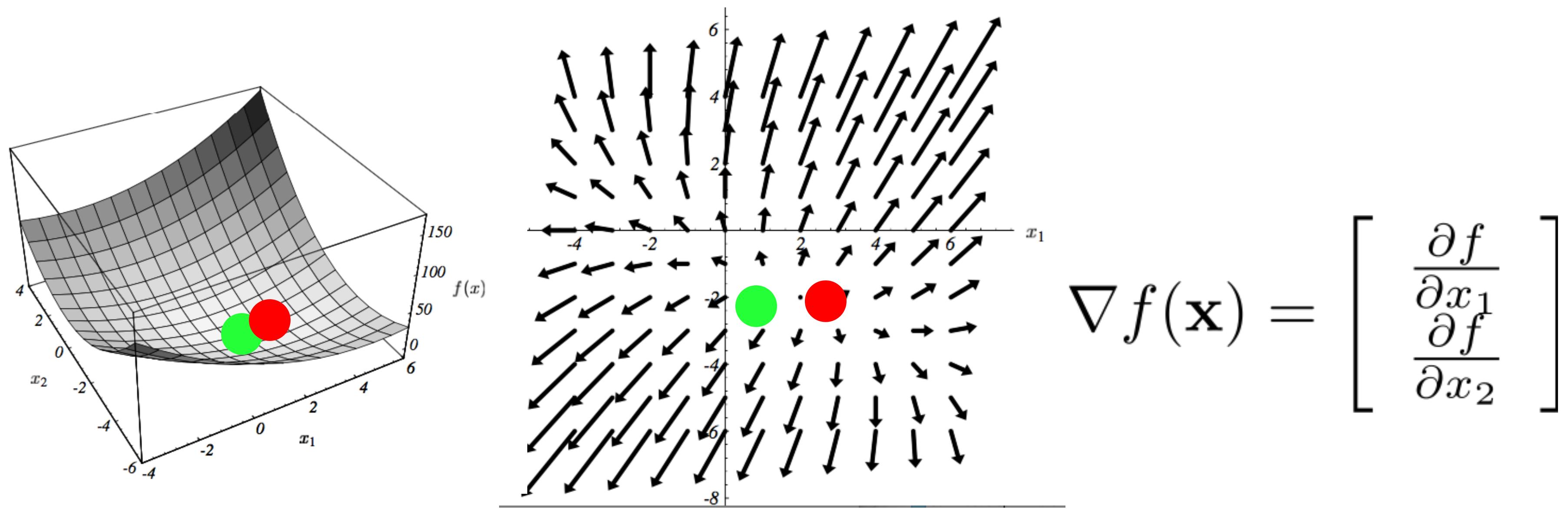
Gradient Descent Minimization



Update:

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

Gradient Descent Minimization



Update:

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