

COMP0169: Machine Learning for Visual Computing

PCA and Linear Classifiers



Lectures will be Recorded

Principal Component Analysis

Principal Component Analysis

- **Problem setup** $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^p$

Principal Component Analysis

- **Problem setup** $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^p$
- **(Optional) Scale variables to unit length**

Principal Component Analysis

- **Problem setup** $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^p$
- **(Optional) Scale variables to unit length**
- **Mean centering** $\mathbf{x}_i \leftarrow \mathbf{x}_i - \bar{\mathbf{x}}$

Principal Component Analysis

- **Problem setup** $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^p$
- **(Optional) Scale variables to unit length**
- **Mean centering** $\mathbf{x}_i \leftarrow \mathbf{x}_i - \bar{\mathbf{x}}$
- **Covariance matrix** $C = X^T X$

Principal Component Analysis

- **Problem setup** $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^p$
- **(Optional) Scale variables to unit length**
- **Mean centering** $\mathbf{x}_i \leftarrow \mathbf{x}_i - \bar{\mathbf{x}}$
- **Covariance matrix** $C = X^T X$
- **Eigen analysis** $C = Q \Lambda Q^T$

Principal Component Analysis

- **Problem setup** $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^p$
- **(Optional) Scale variables to unit length**
- **Mean centering** $\mathbf{x}_i \leftarrow \mathbf{x}_i - \bar{\mathbf{x}}$
- **Covariance matrix** $C = X^T X$
- **Eigen analysis** $C = Q \Lambda Q^T$
- **Or, SVD** $X = U \Sigma W^T$
 $C = X^T X = W \Sigma^2 W^T$

Principal Component Analysis

$$\mathbf{y} = \bar{\mathbf{x}} + \sum_{i=1}^p \beta_i \mathbf{e}_i$$

$$\beta_i = \langle \mathbf{y} - \bar{\mathbf{x}}, \mathbf{e}_i \rangle = (\mathbf{y} - \bar{\mathbf{x}})^T \mathbf{e}_i$$

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

Bias Trick

$$y = w_1x_1 + w_2x_2 + \dots w_px_p + b$$

Bias Trick

$$\begin{aligned}y &= w_1x_1 + w_2x_2 + \dots w_px_p + b \\&= b \cdot 1 + w_1x_1 + w_2x_2 + \dots w_px_p\end{aligned}$$

Bias Trick

$$\begin{aligned}y &= w_1x_1 + w_2x_2 + \dots w_px_p + b \\&= b \cdot 1 + w_1x_1 + w_2x_2 + \dots w_px_p \\&= \langle [b \ w_1 \ \dots \ w_p], [1 \ x_1 \ \dots \ x_p] \rangle\end{aligned}$$

Bias Trick

$$\begin{aligned}y &= w_1x_1 + w_2x_2 + \dots w_px_p + b \\&= b \cdot 1 + w_1x_1 + w_2x_2 + \dots w_px_p \\&= \langle [b \ w_1 \ \dots \ w_p], [1 \ x_1 \ \dots \ x_p] \rangle \\&= \langle \mathbf{w}, \mathbf{x} \rangle\end{aligned}$$

Bias Trick

$$\begin{aligned}y &= w_1x_1 + w_2x_2 + \dots w_px_p + b \\&= b \cdot 1 + w_1x_1 + w_2x_2 + \dots w_px_p \\&= \langle [b \ w_1 \ \dots \ w_p], [1 \ x_1 \ \dots \ x_p] \rangle \\&= \langle \mathbf{w}, \mathbf{x} \rangle \\&= \mathbf{w}^T \mathbf{x}\end{aligned}$$

Loss function and Optimization

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^i - y^i)$$

Loss function and Optimization

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^i - y^i)$$

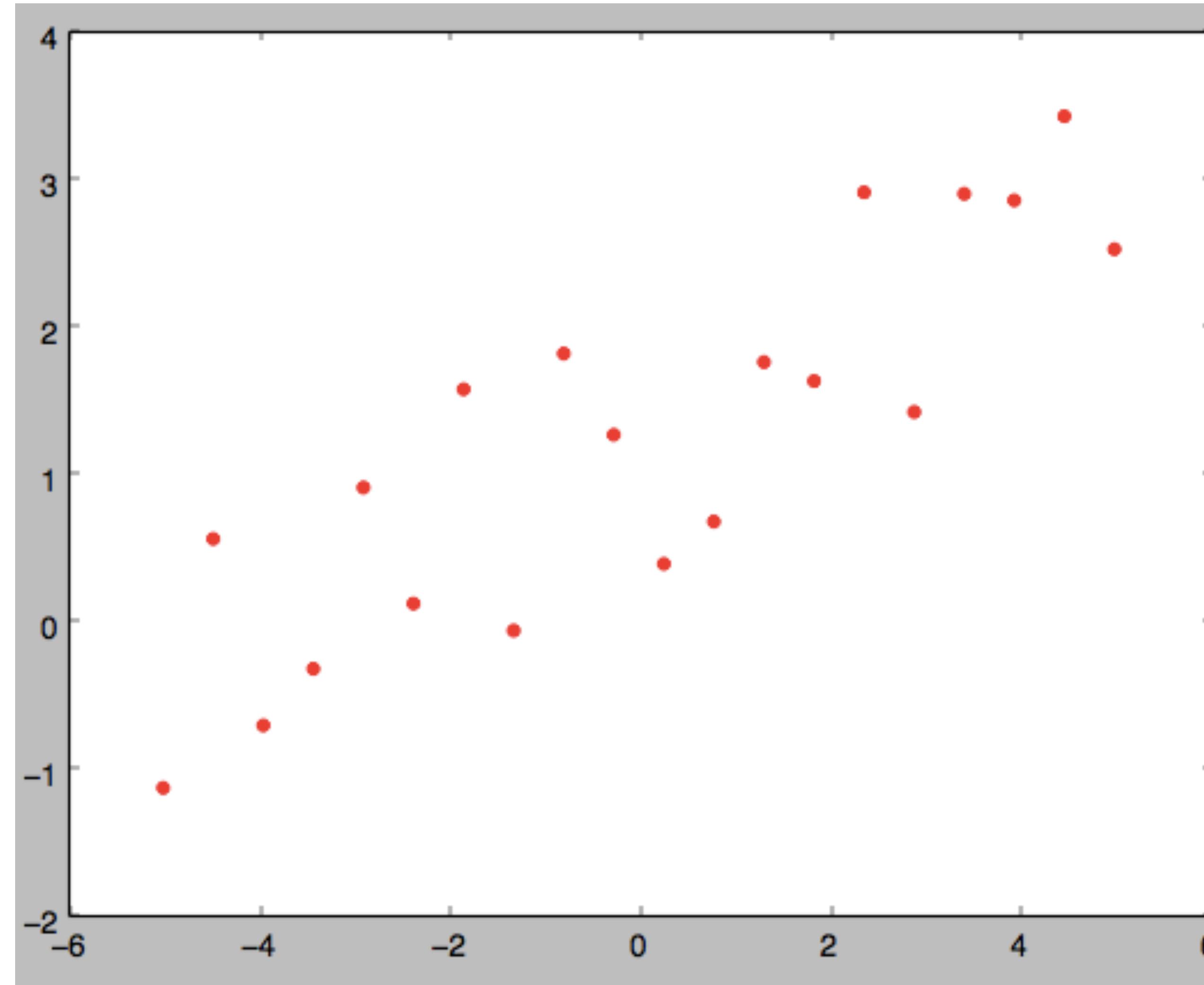
$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

Loss function and Optimization

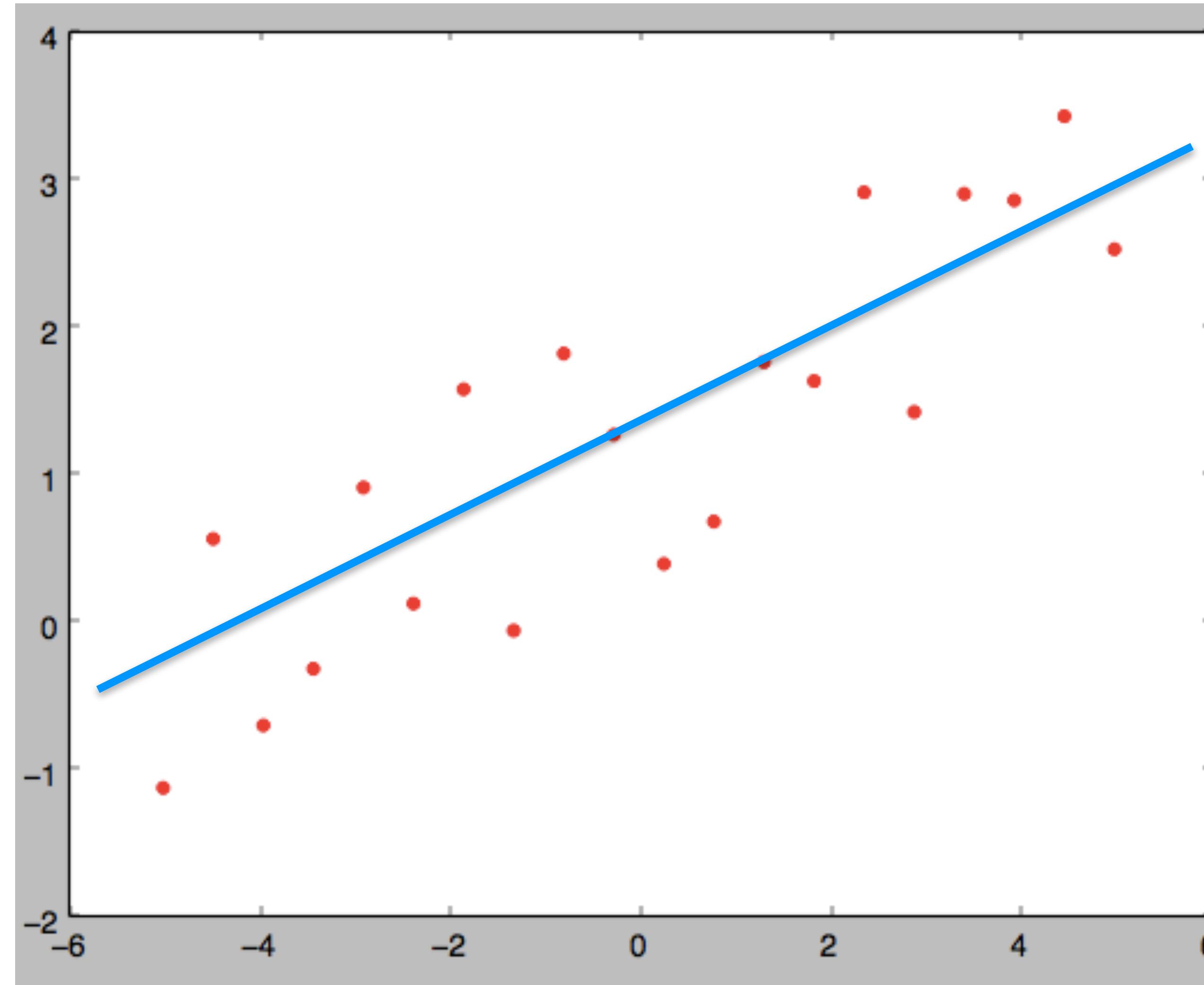
$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}^i - y^i)$$

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = 0$$

Regression: Continuous Output



Regression: Continuous Output



Normal Equation (derivation)

Line Fitting

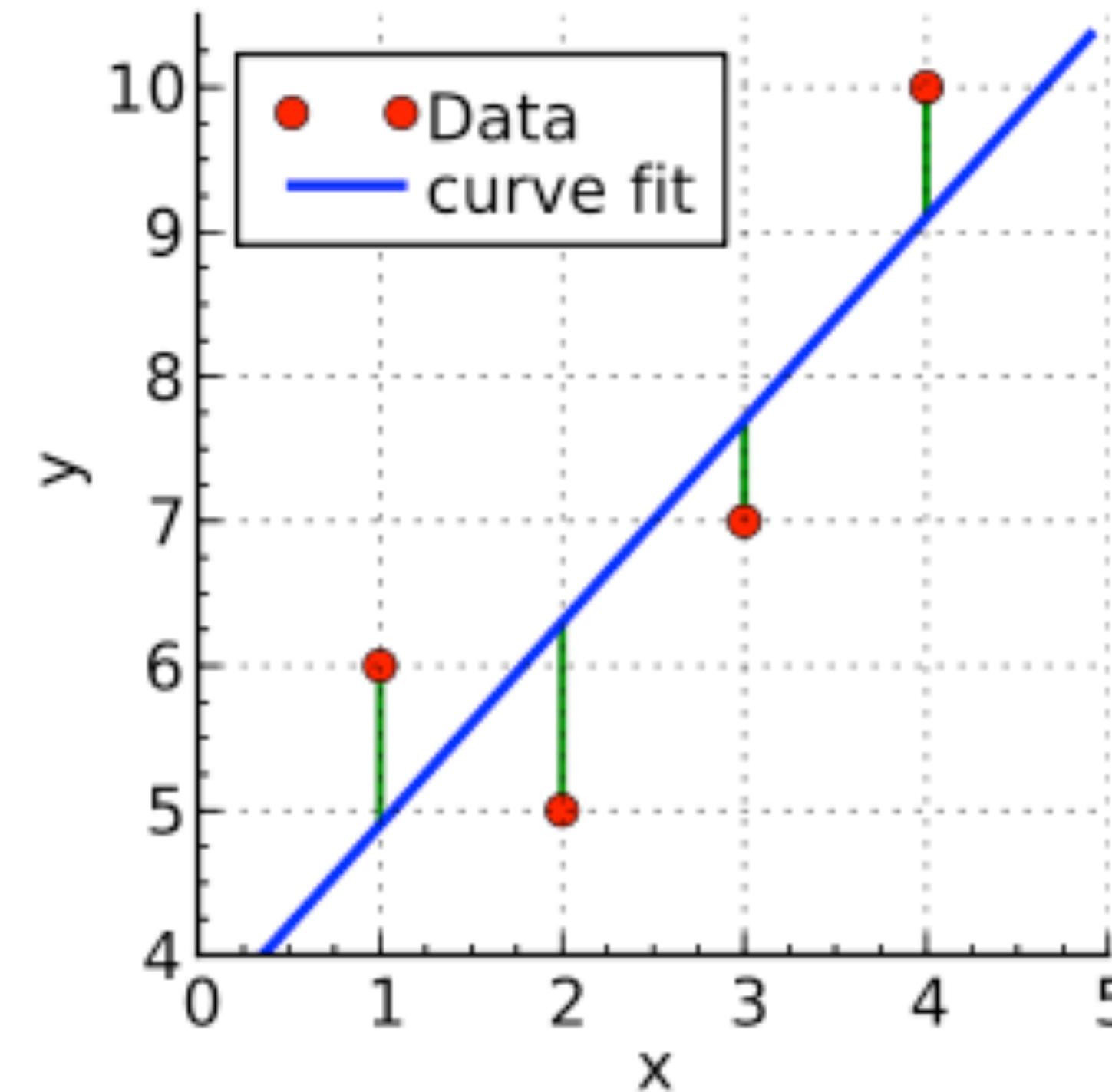
$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Line Fitting

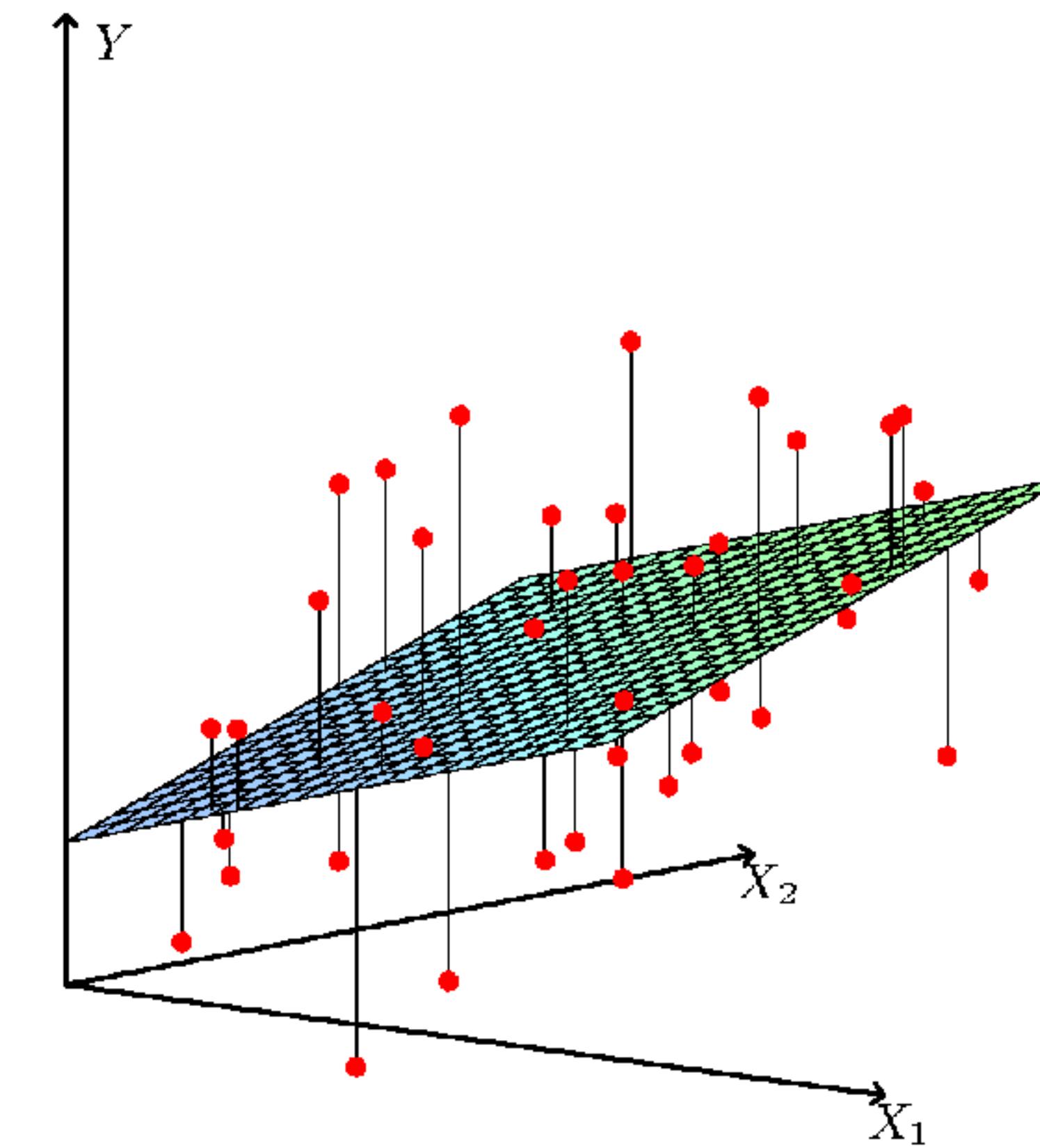
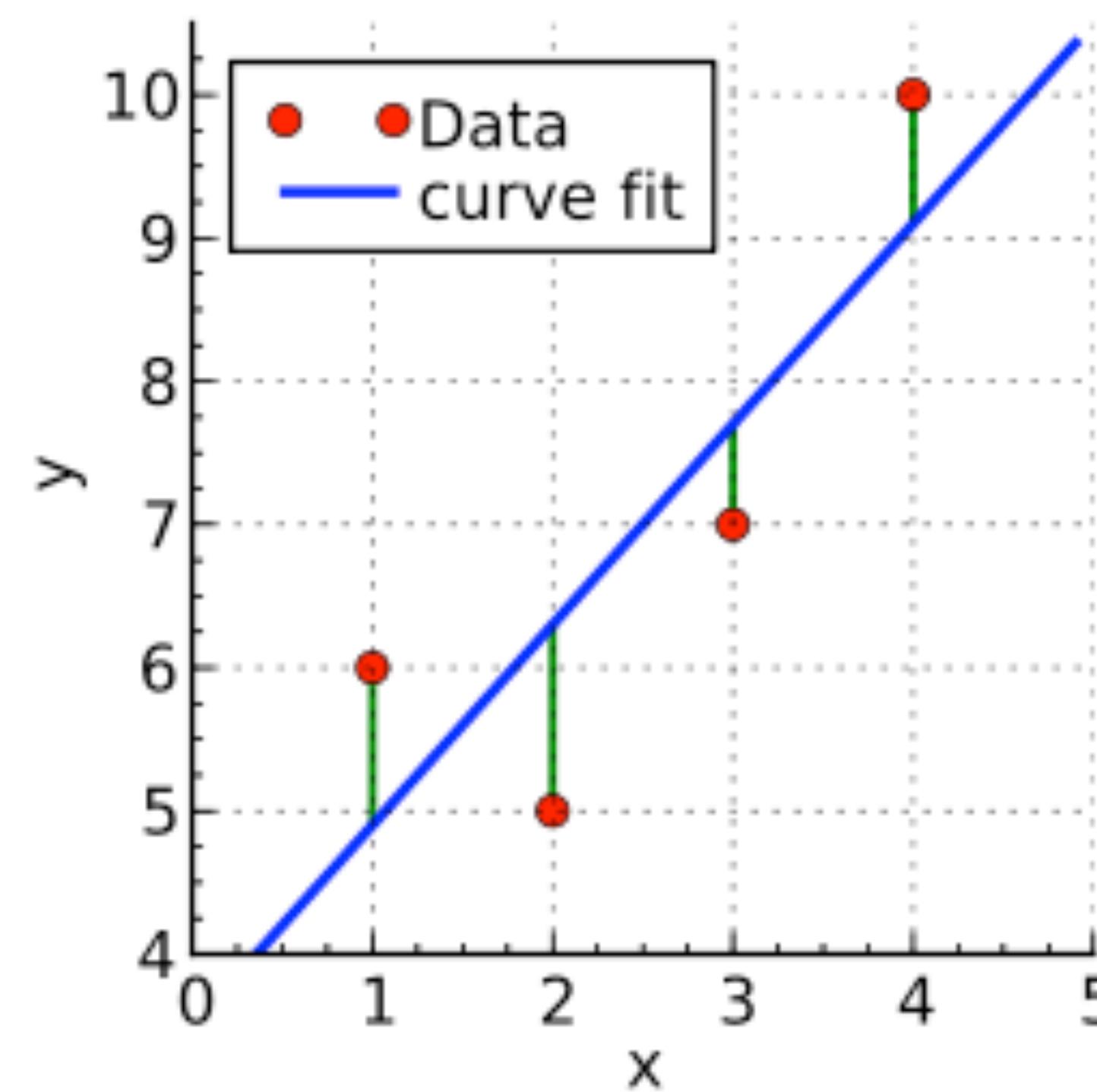
$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

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

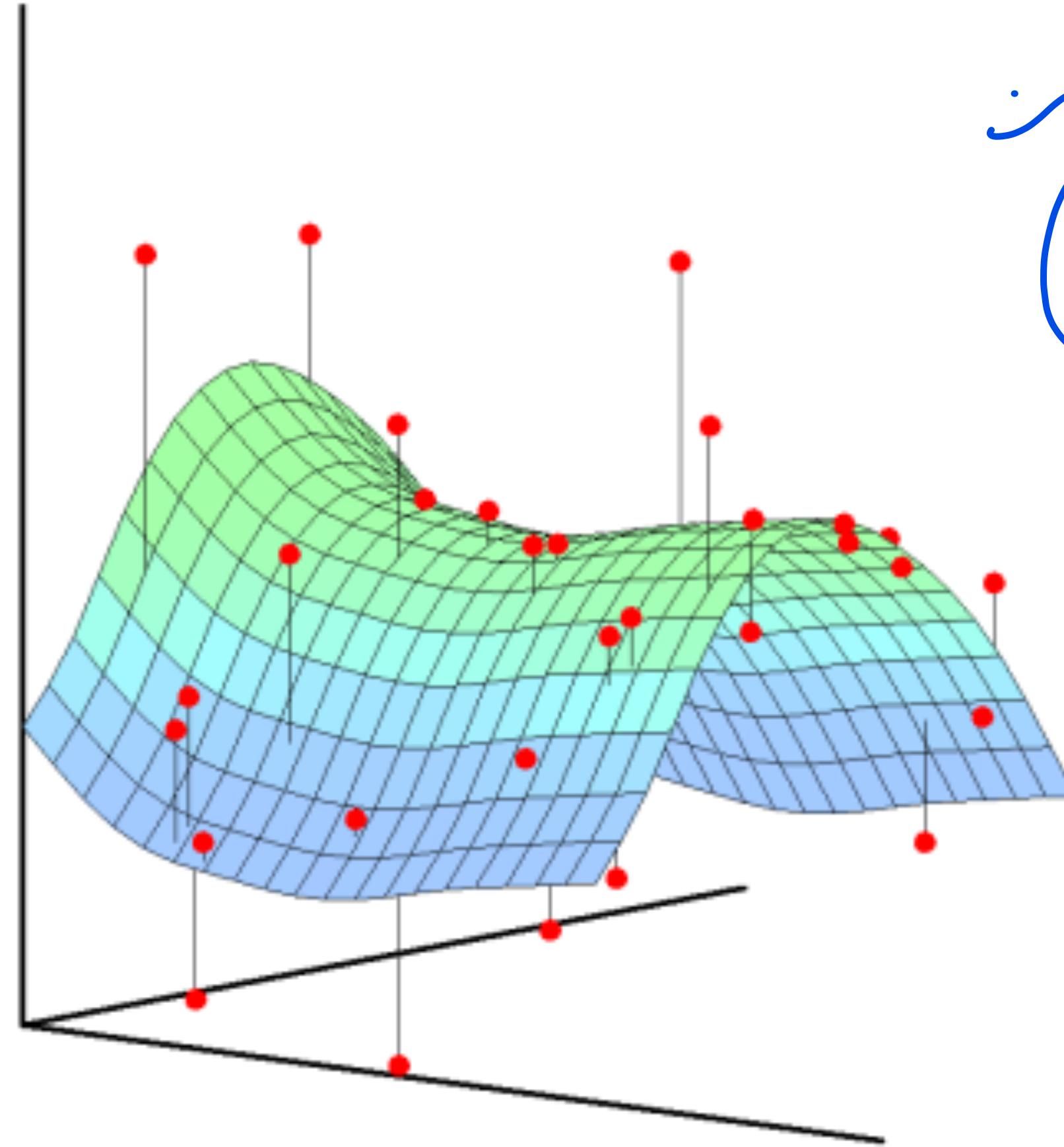
Linear Regression (Line/Plane Fitting)



Linear Regression (Line/Plane Fitting)



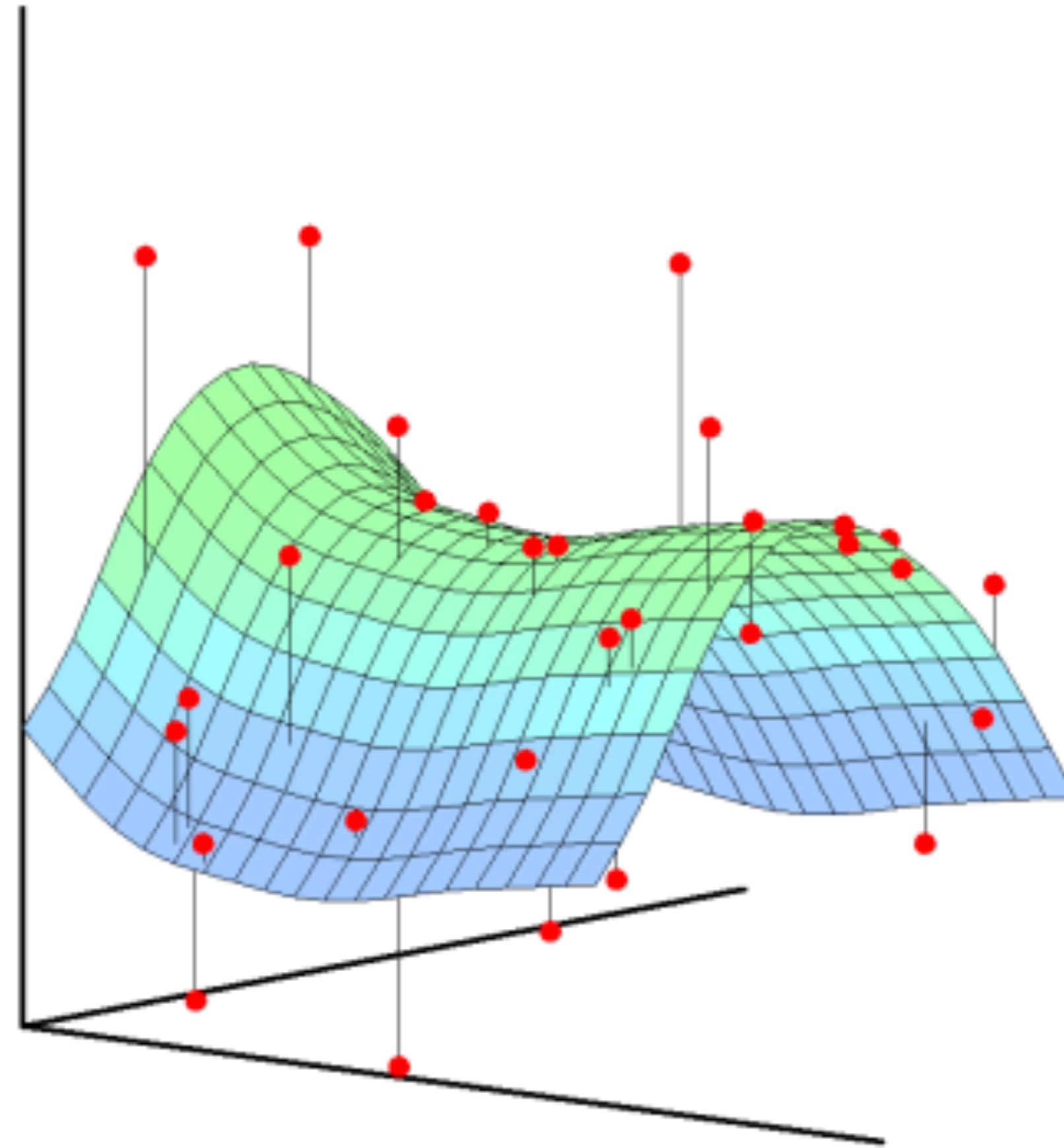
Polynomial Linear Regression



$$y = w_0 x_0 + w_1 x_1 + w_2 x_0^2 + w_3 x_0 x_1 + w_4 x_1^2$$

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

Polynomial Linear Regression

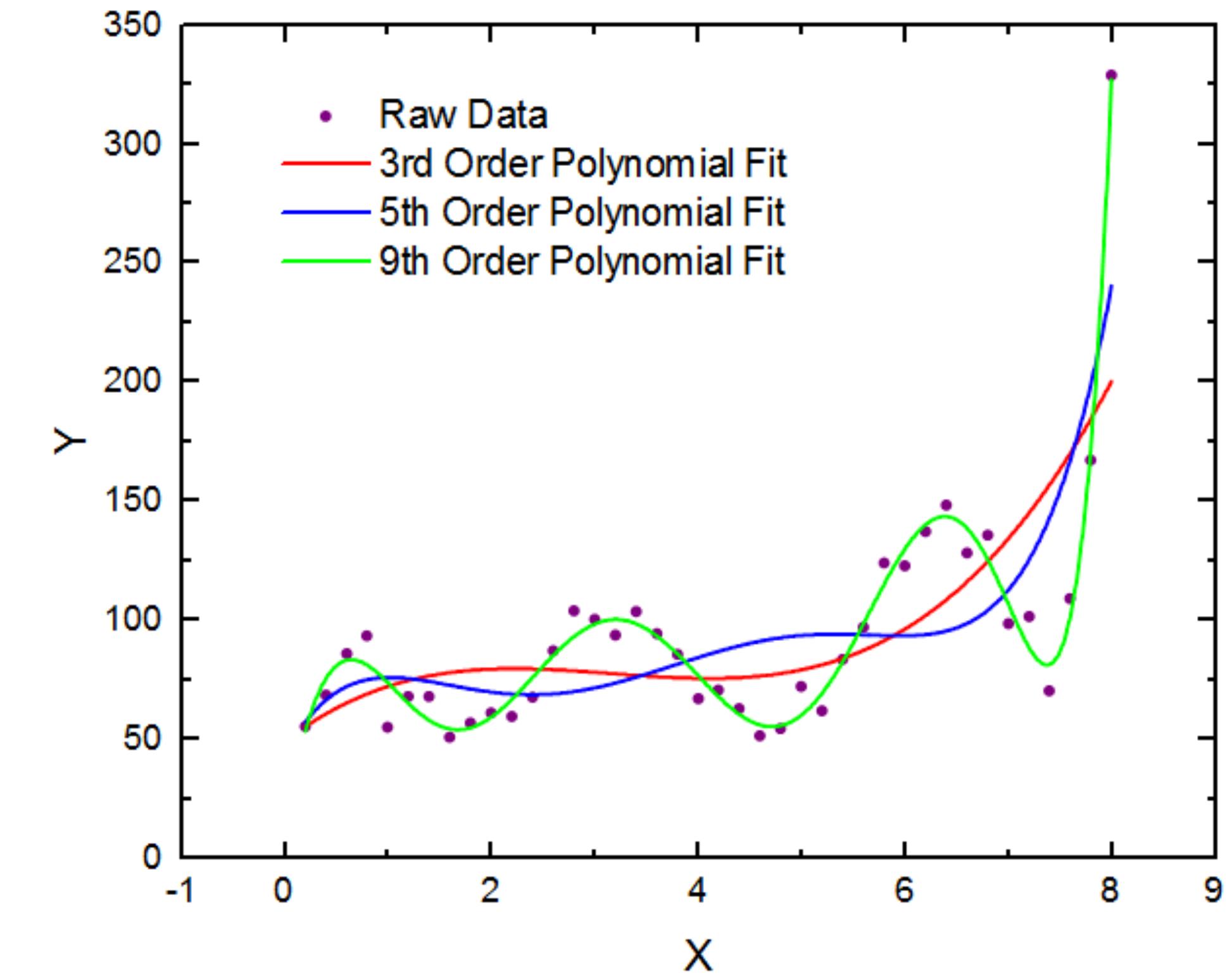


$$\mathbf{x} \rightarrow \phi(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}$$

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

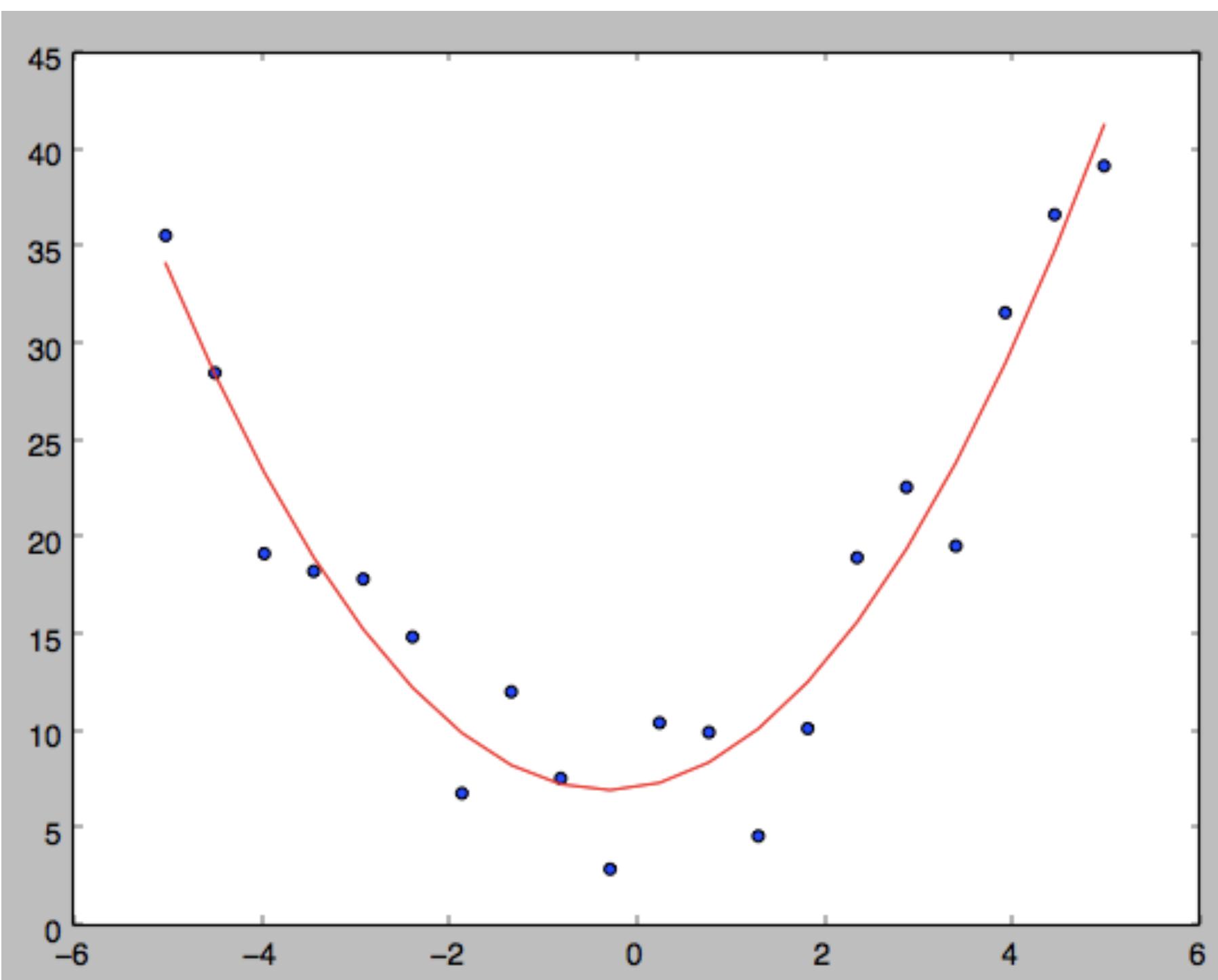
$$\mathbf{y} = \Phi \mathbf{w} + \boldsymbol{\epsilon}$$

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

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

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

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))

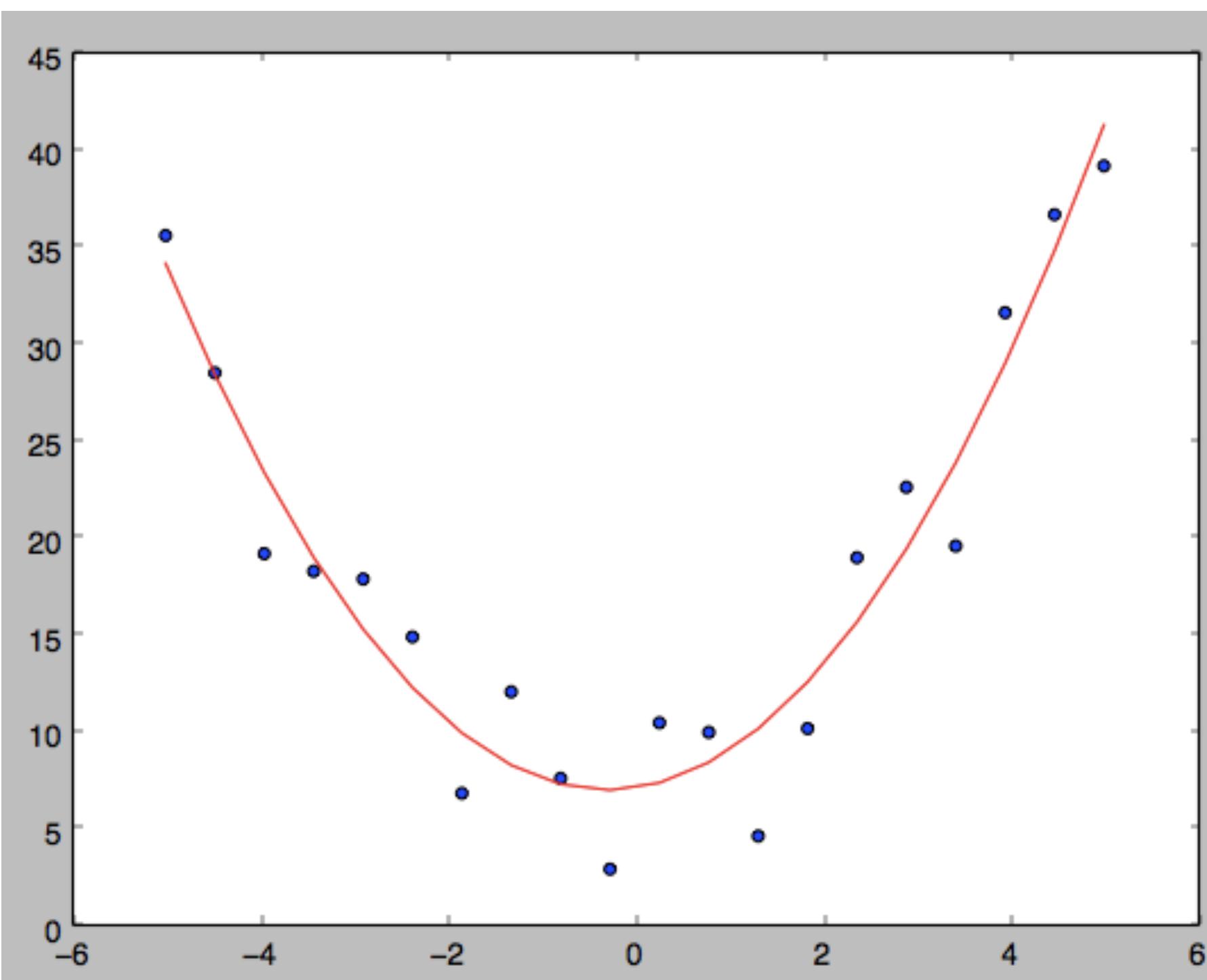
# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1), (x**2).reshape(-1, 1)] )

# 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)

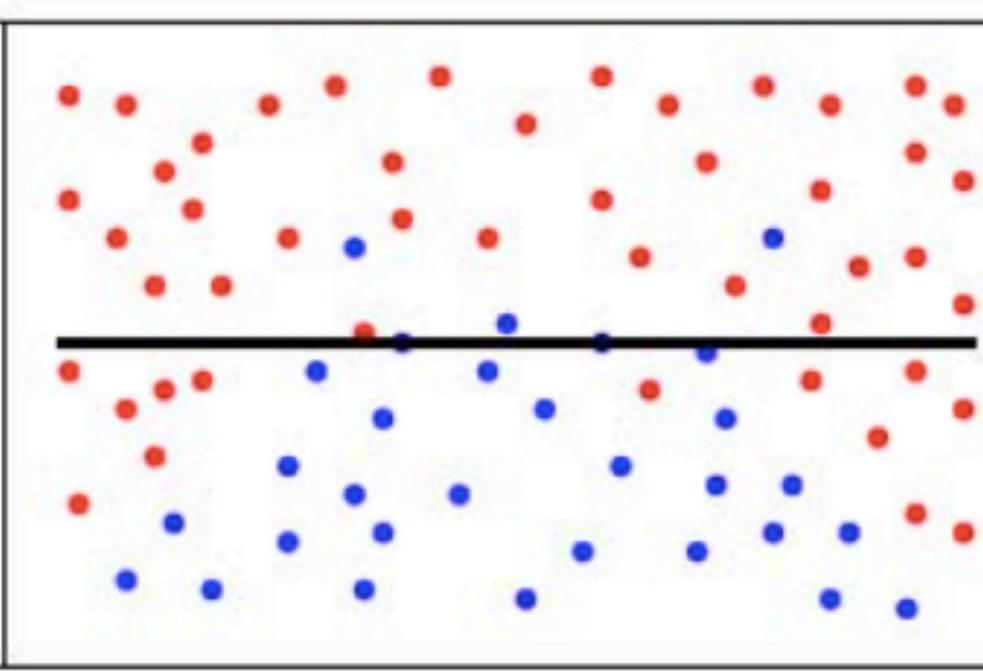
# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x + w_est[2]*x**2

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

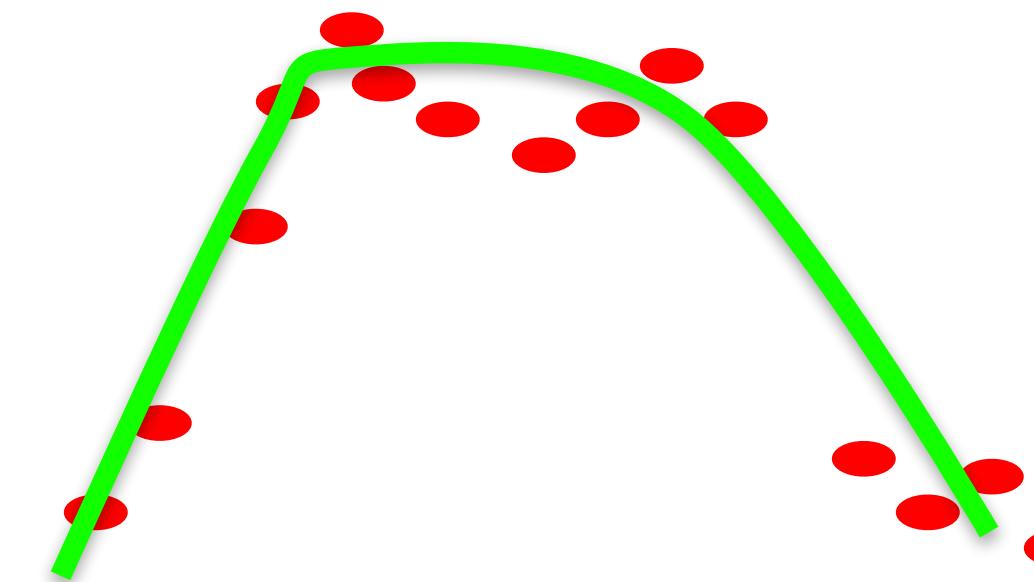
$$\mathbf{w}^* = (\Phi^T \Phi)^{-1} \Phi \mathbf{y}$$

Underfitting vs. Overfitting

classification



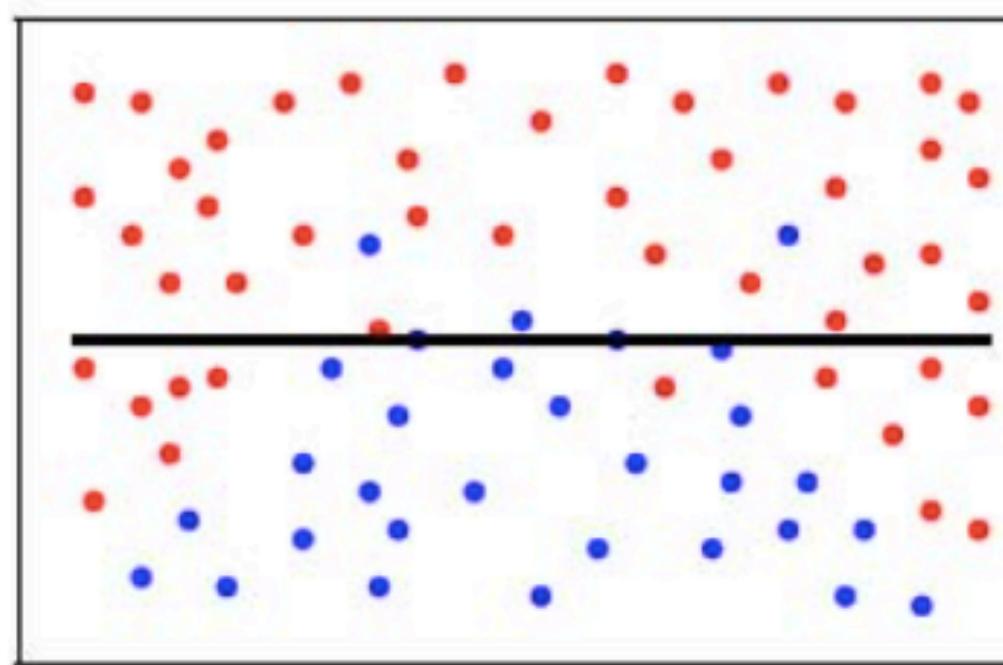
regression



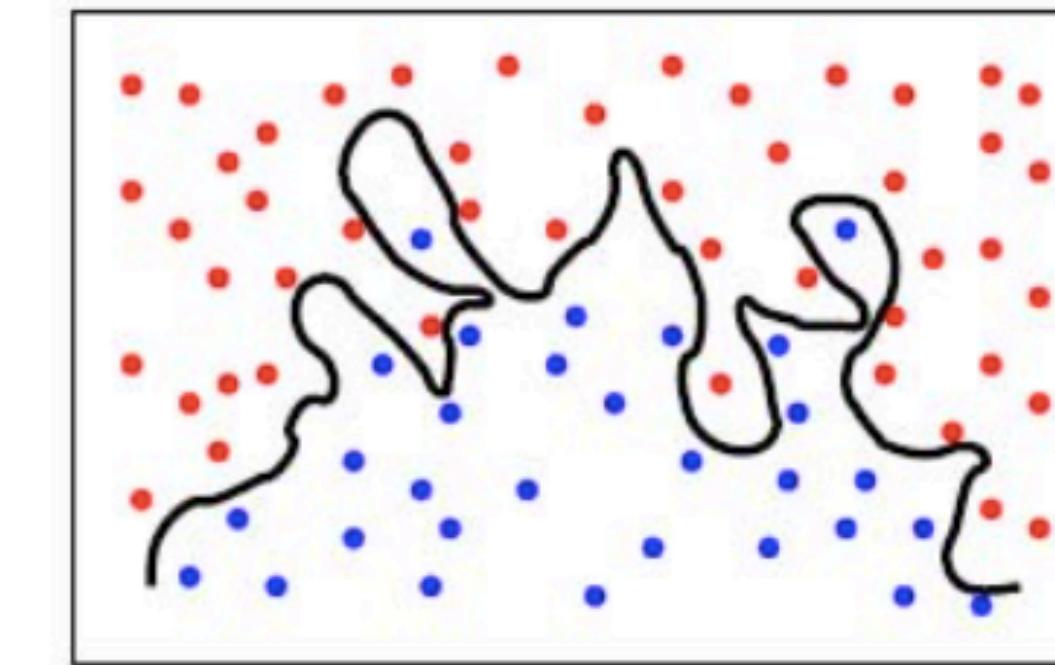
Underfitting vs. Overfitting

classification

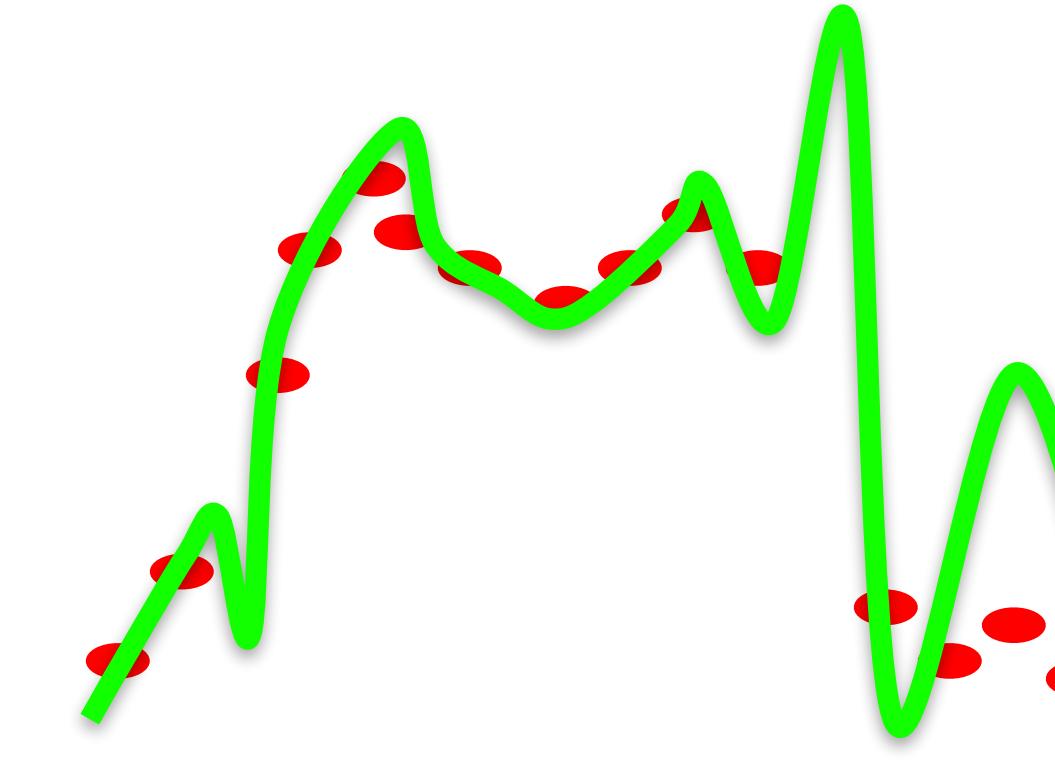
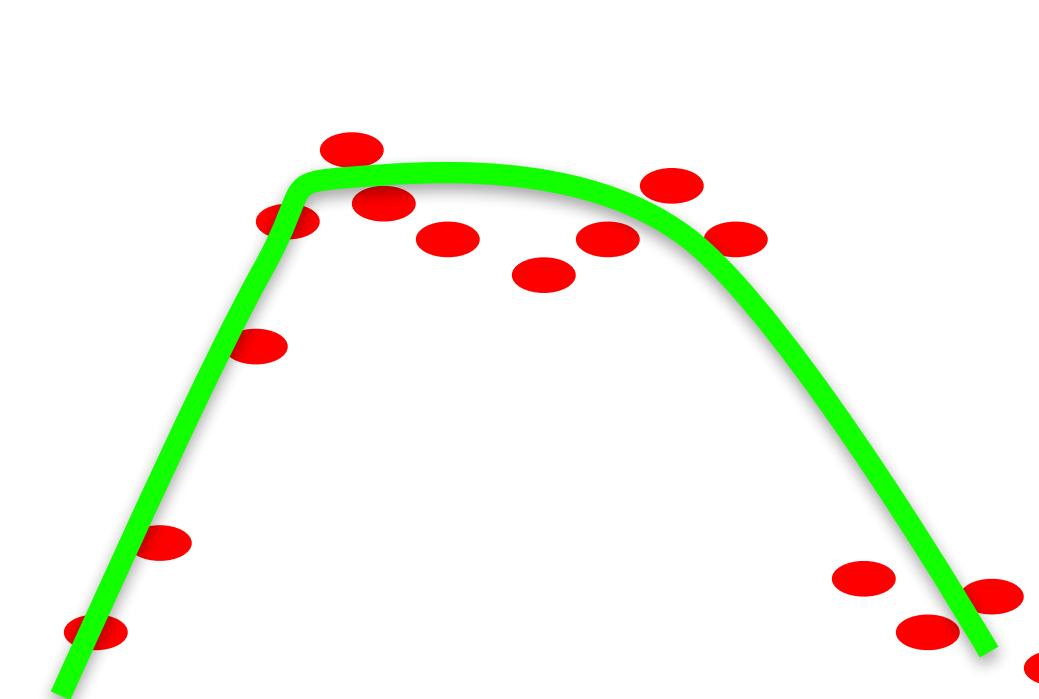
Underfitting



Overfitting



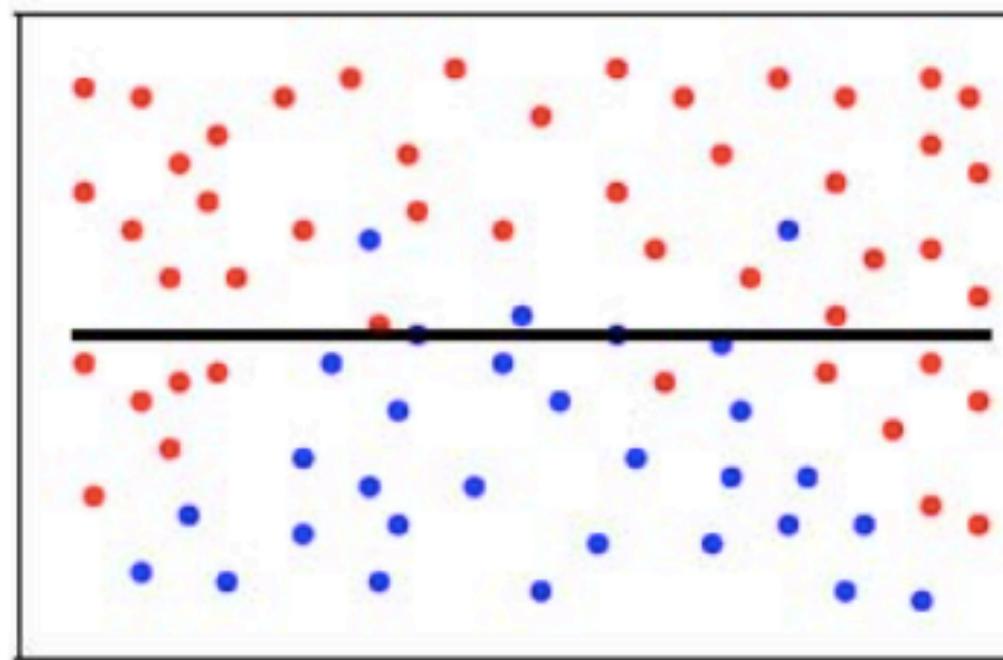
regression



Underfitting vs. Overfitting

classification

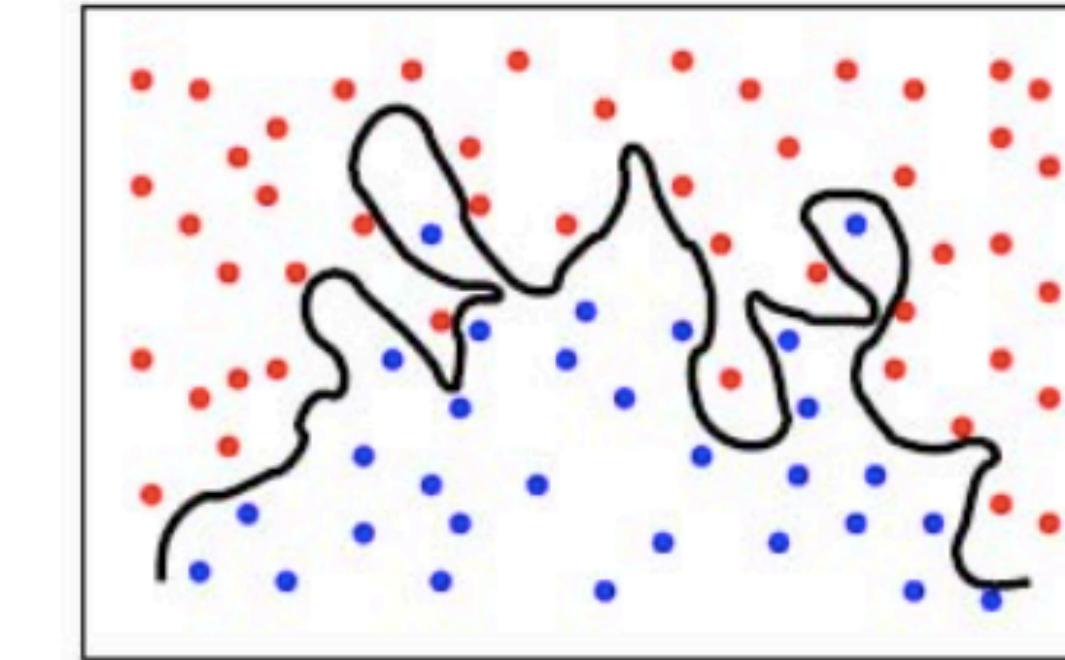
Underfitting



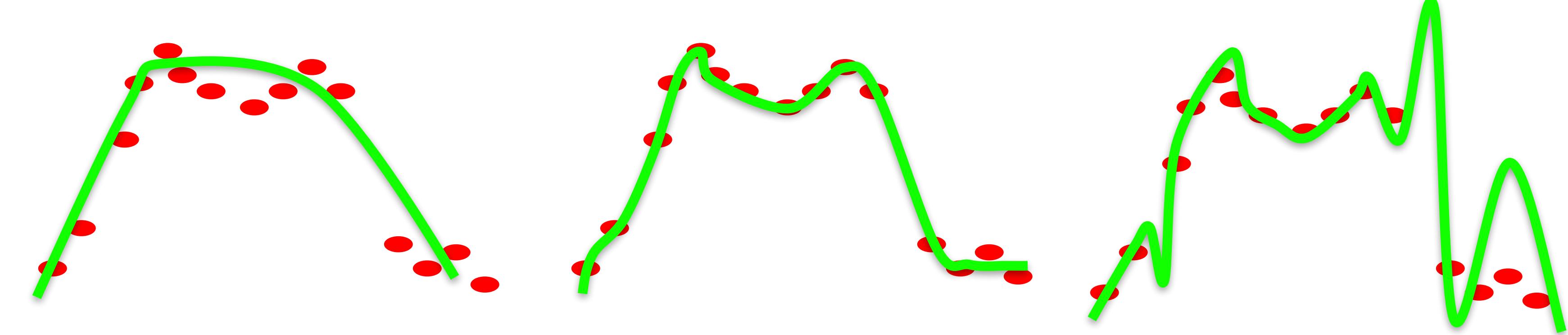
just right



Overfitting



regression



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

Linear Binary Classifier

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C \\ -1 & \mathbf{x}_i \notin C \end{cases}$$

Linear Binary Classifier

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C \\ -1 & \mathbf{x}_i \notin C \end{cases}$$

Find weights w such that

Linear Binary Classifier

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C \\ -1 & \text{if } \mathbf{x}_i \notin C \end{cases}$$

Find weights w such that

$$\begin{aligned} \mathbf{x}_i^T \mathbf{w} \geq 0 & \quad \text{if } y_i = 1 \\ \mathbf{x}_i^T \mathbf{w} < 0 & \quad \text{if } y_i = -1 \end{aligned}$$

Linear Binary Classifier

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C \\ -1 & \text{if } \mathbf{x}_i \notin C \end{cases}$$

Find weights w such that

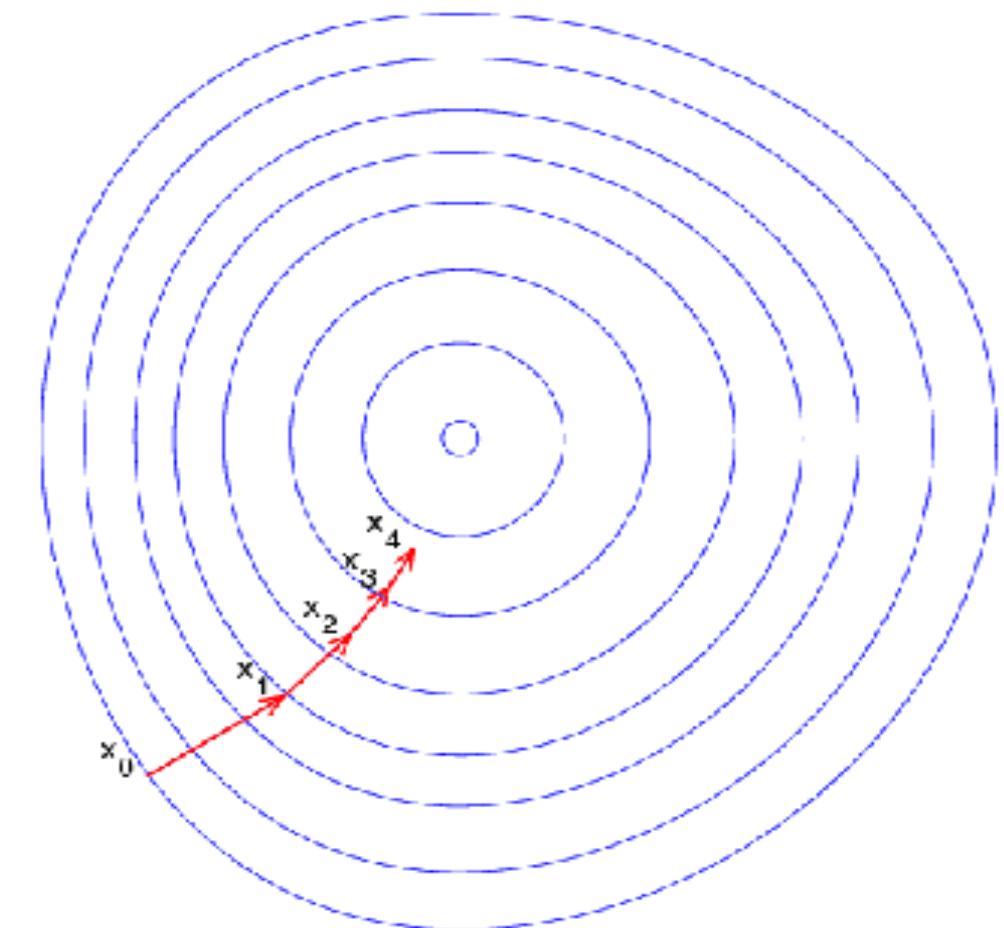
$$\begin{aligned} \mathbf{x}_i^T \mathbf{w} \geq 0 & \quad \text{if } y_i = 1 \\ \mathbf{x}_i^T \mathbf{w} < 0 & \quad \text{if } y_i = -1 \end{aligned}$$

$$y_i (\mathbf{x}_i^T \mathbf{w}) \geq 0$$

Gradient Descent Minimization

Initialize: \mathbf{x}_0

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

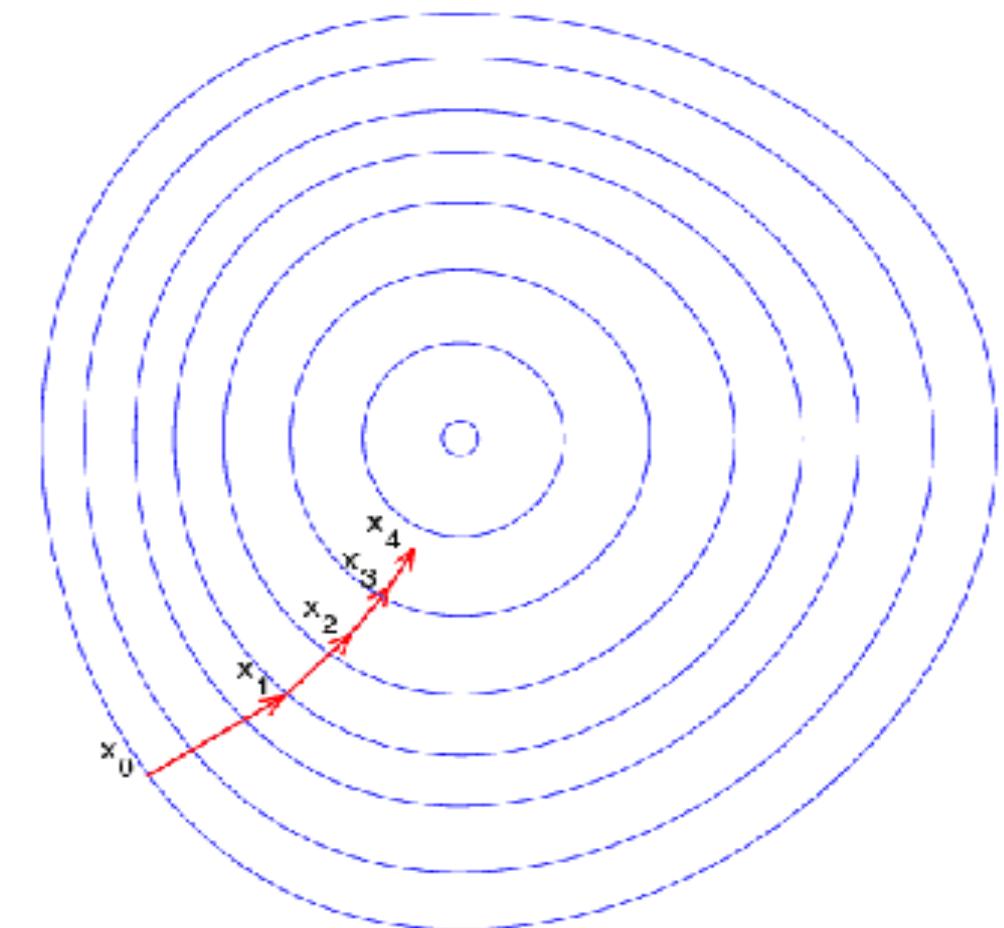


Gradient Descent Minimization

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.

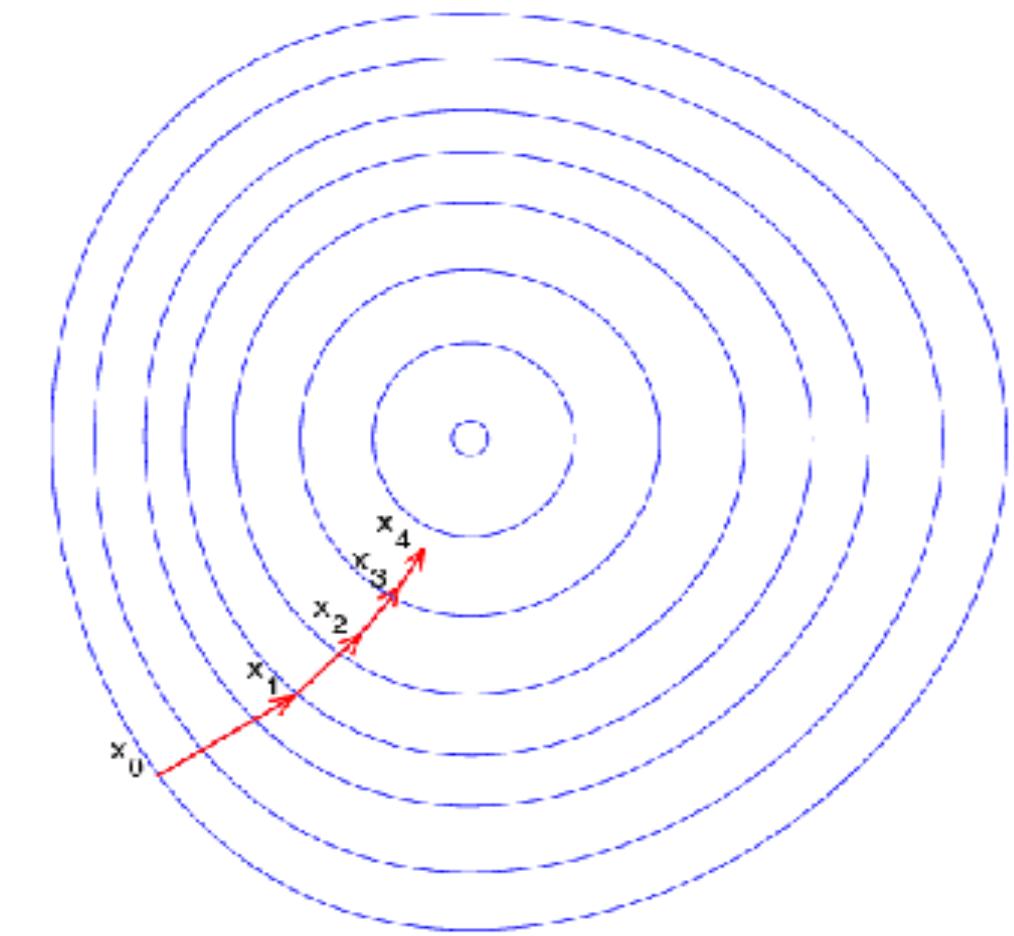
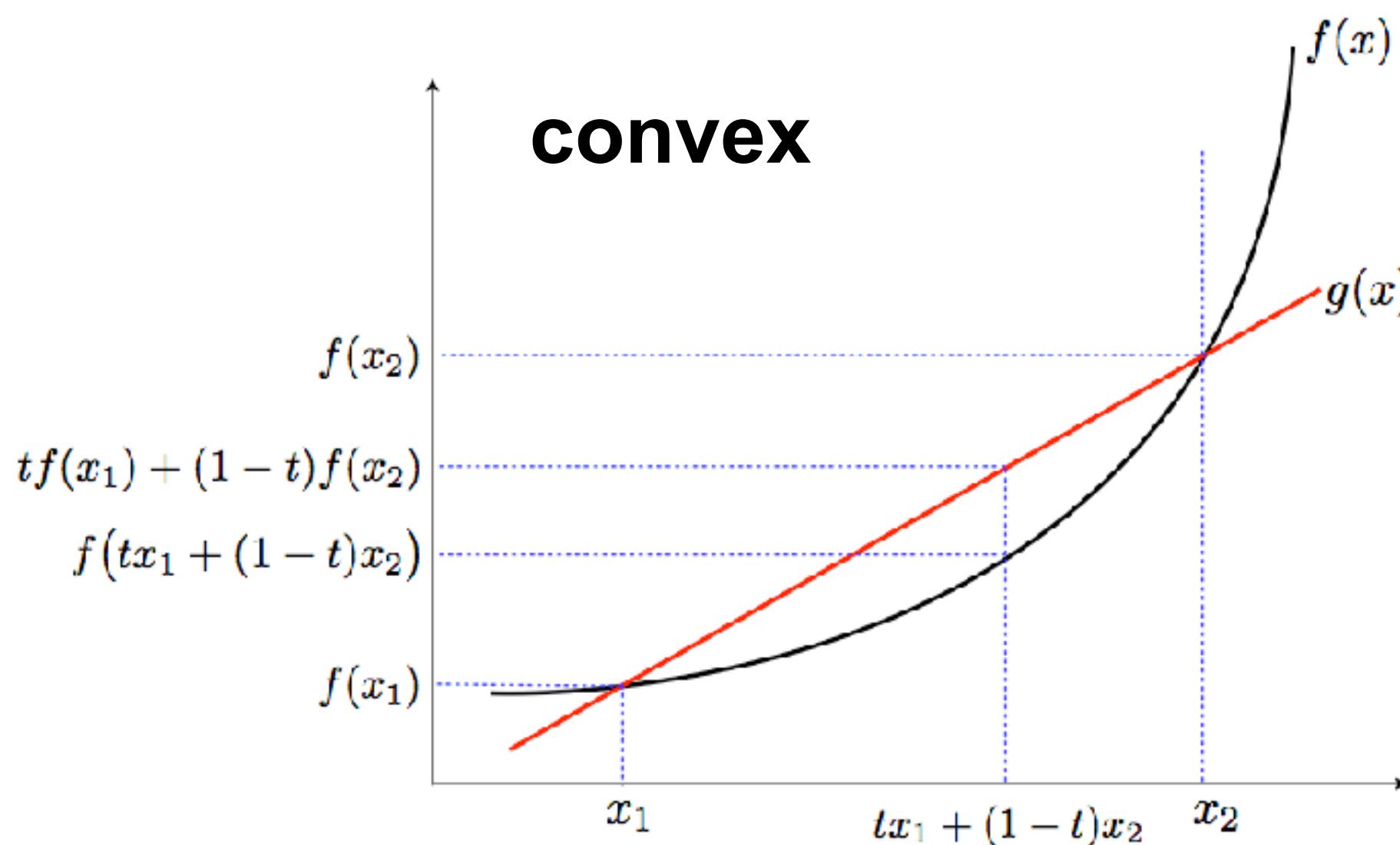


Gradient Descent Minimization

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.

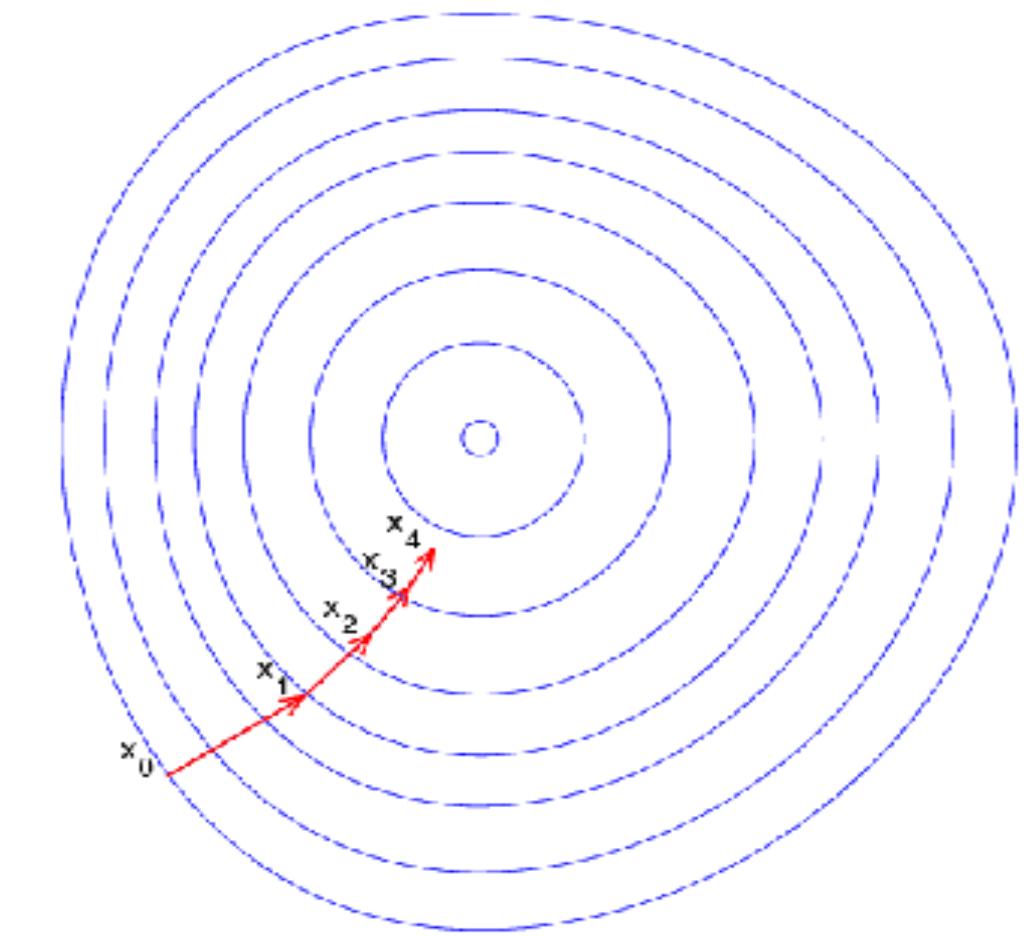
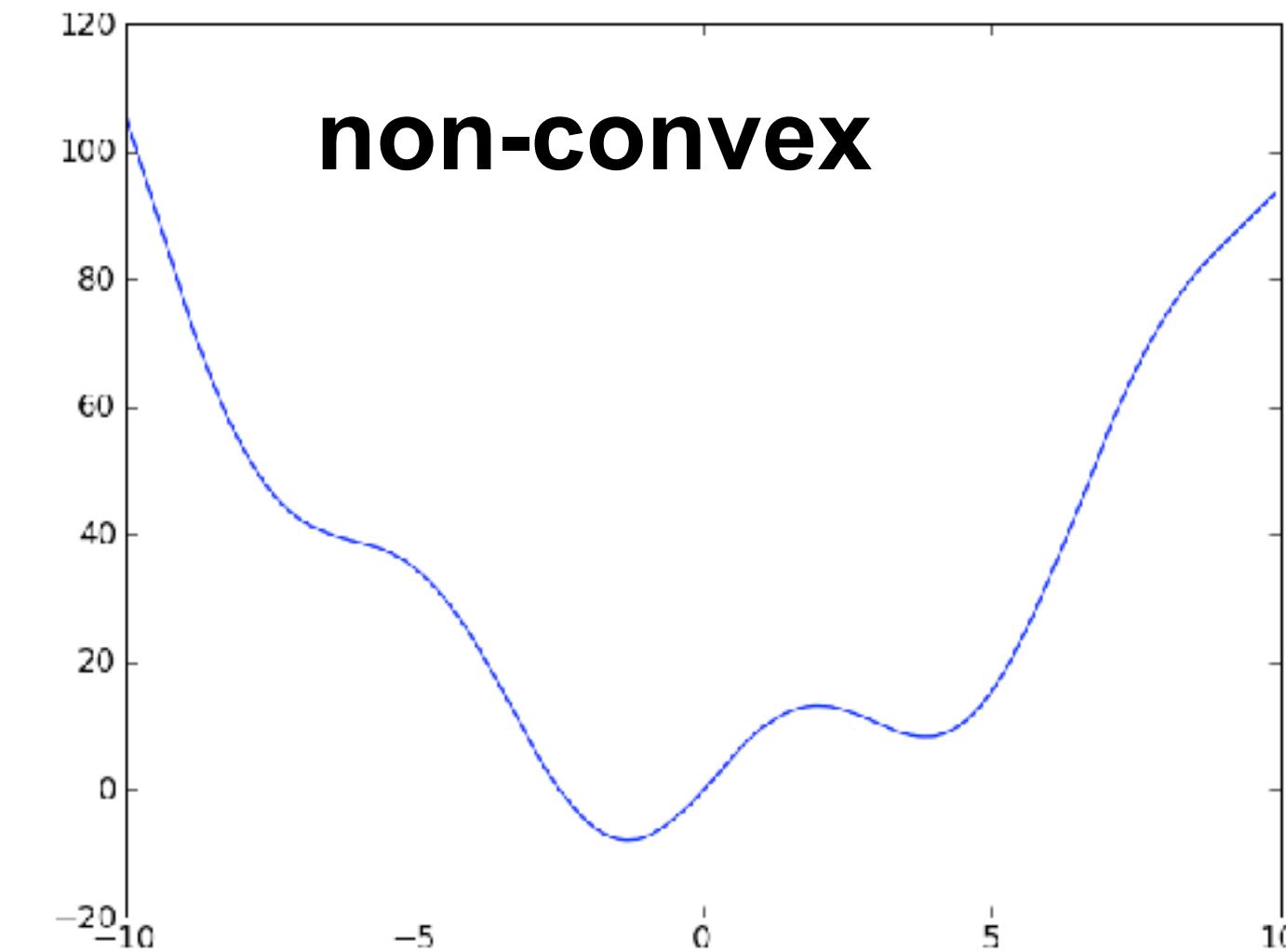
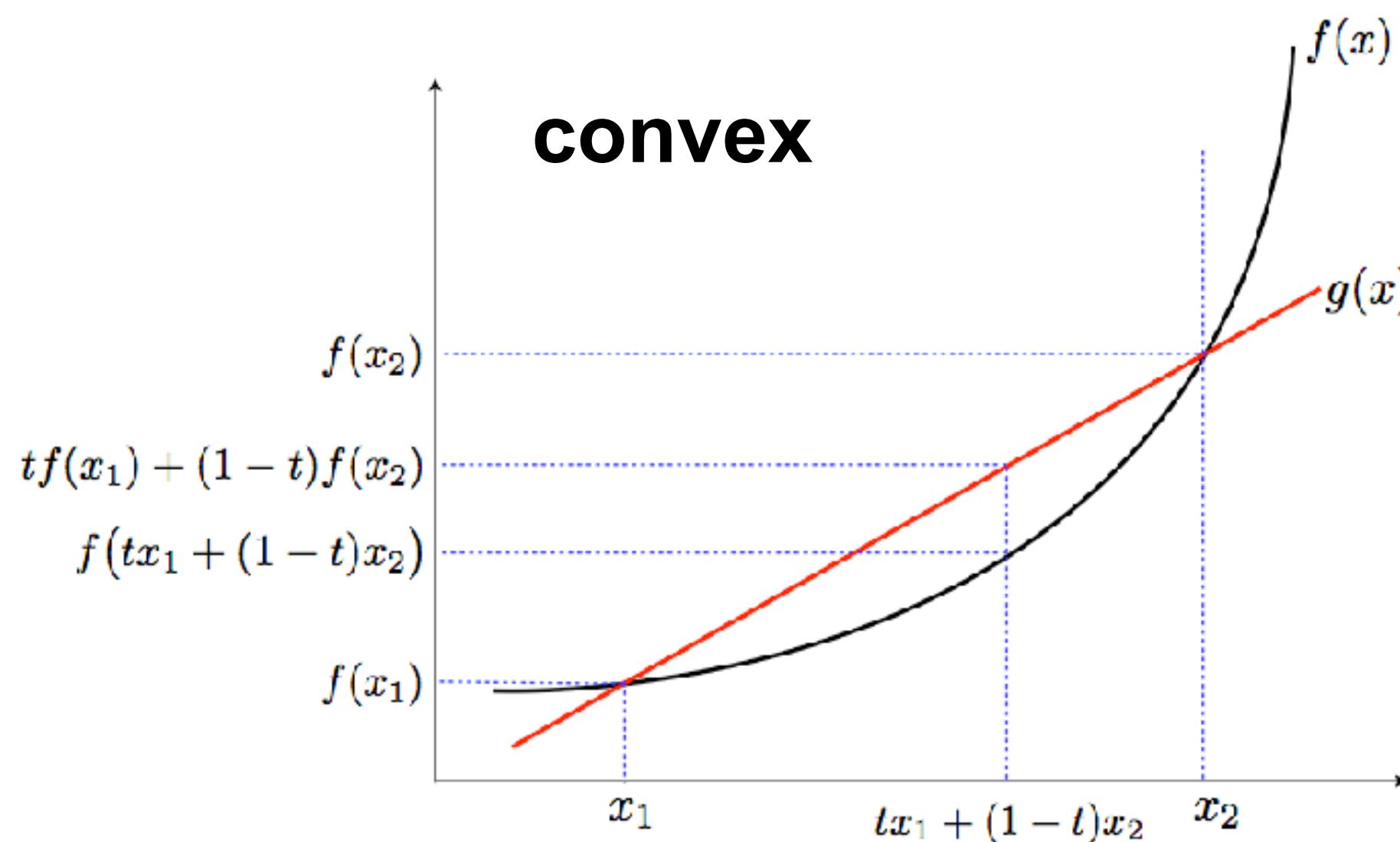


Image Classifier

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)**

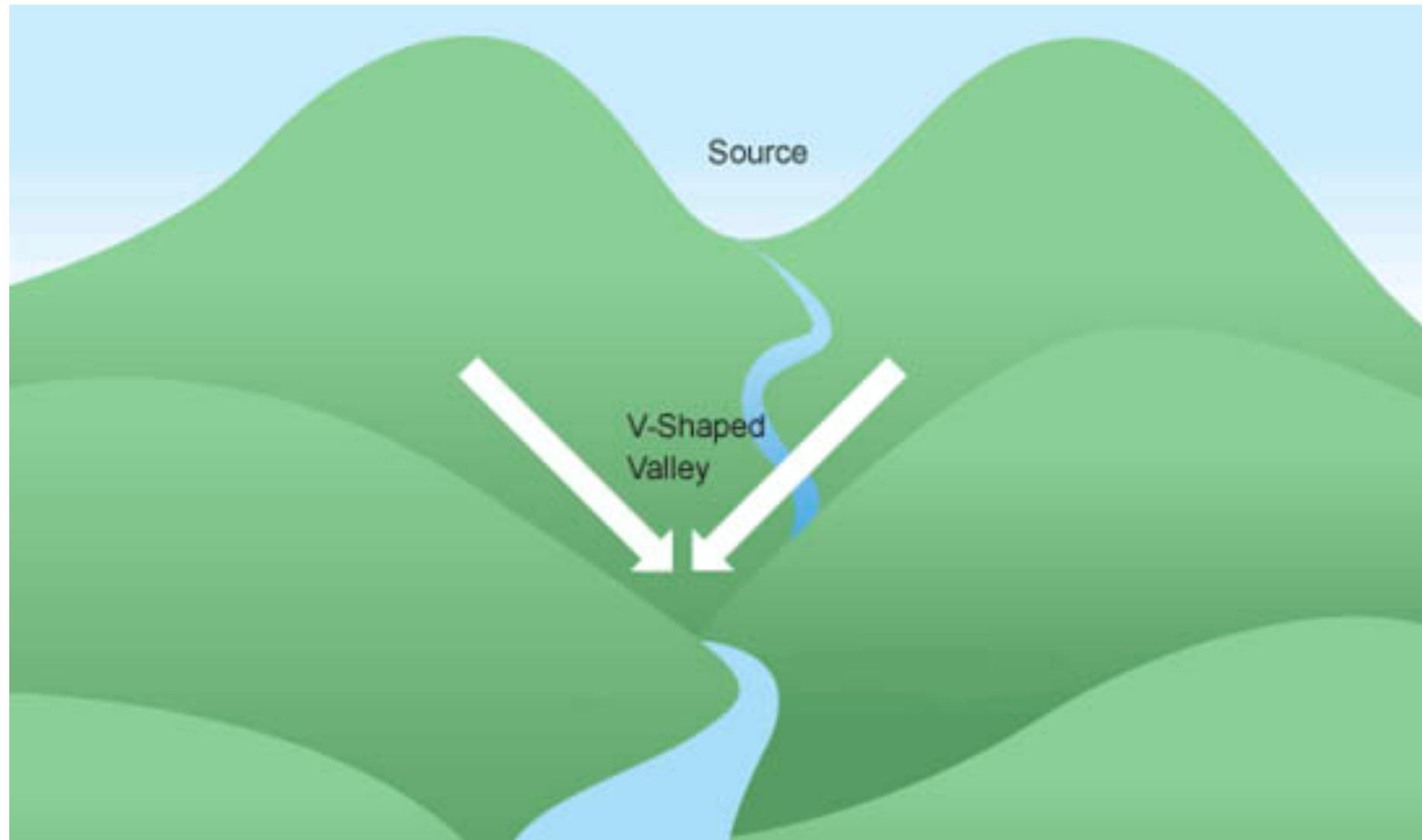
$$f(\mathbf{x}_i, \mathbf{W})$$

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

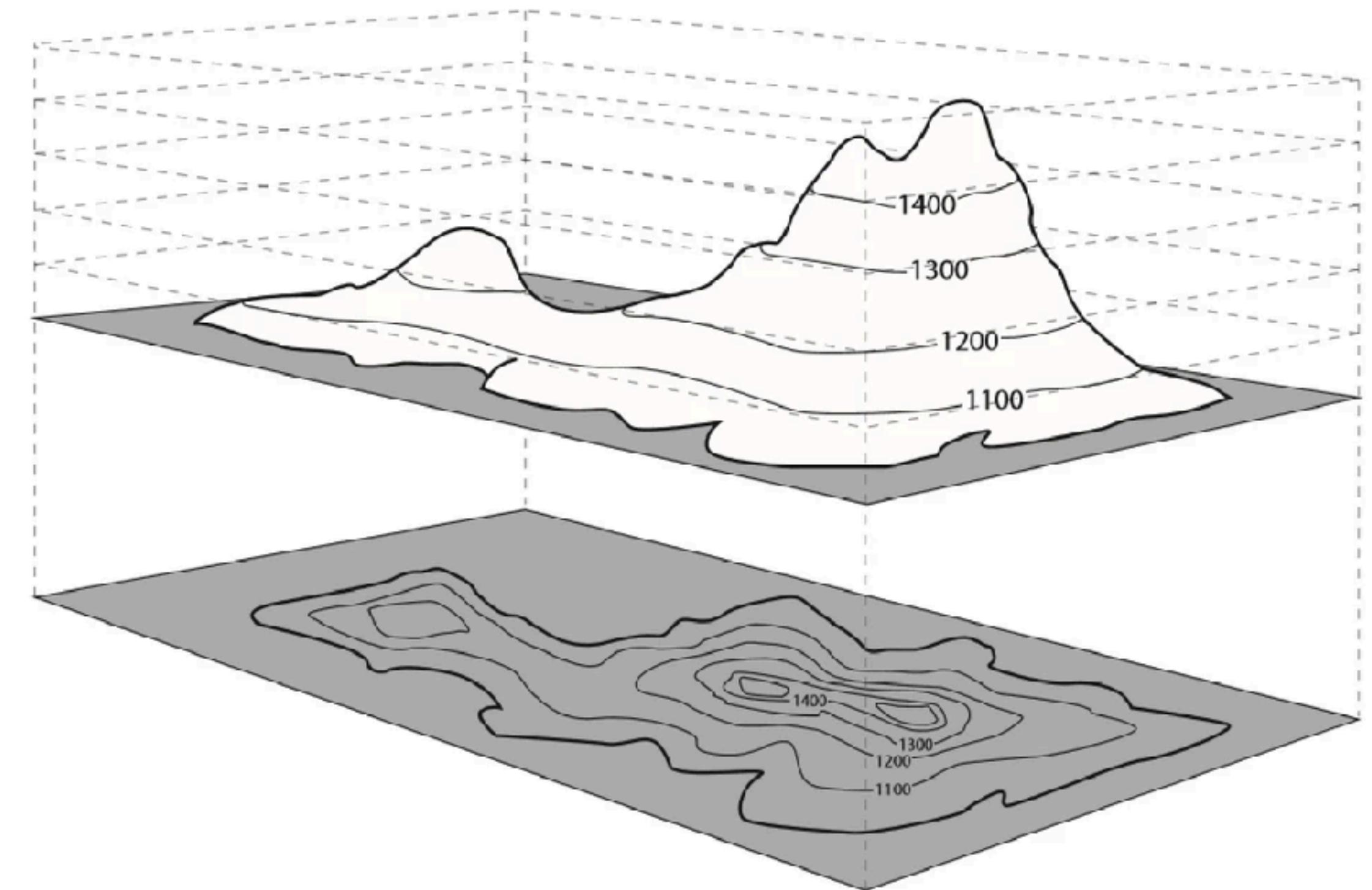
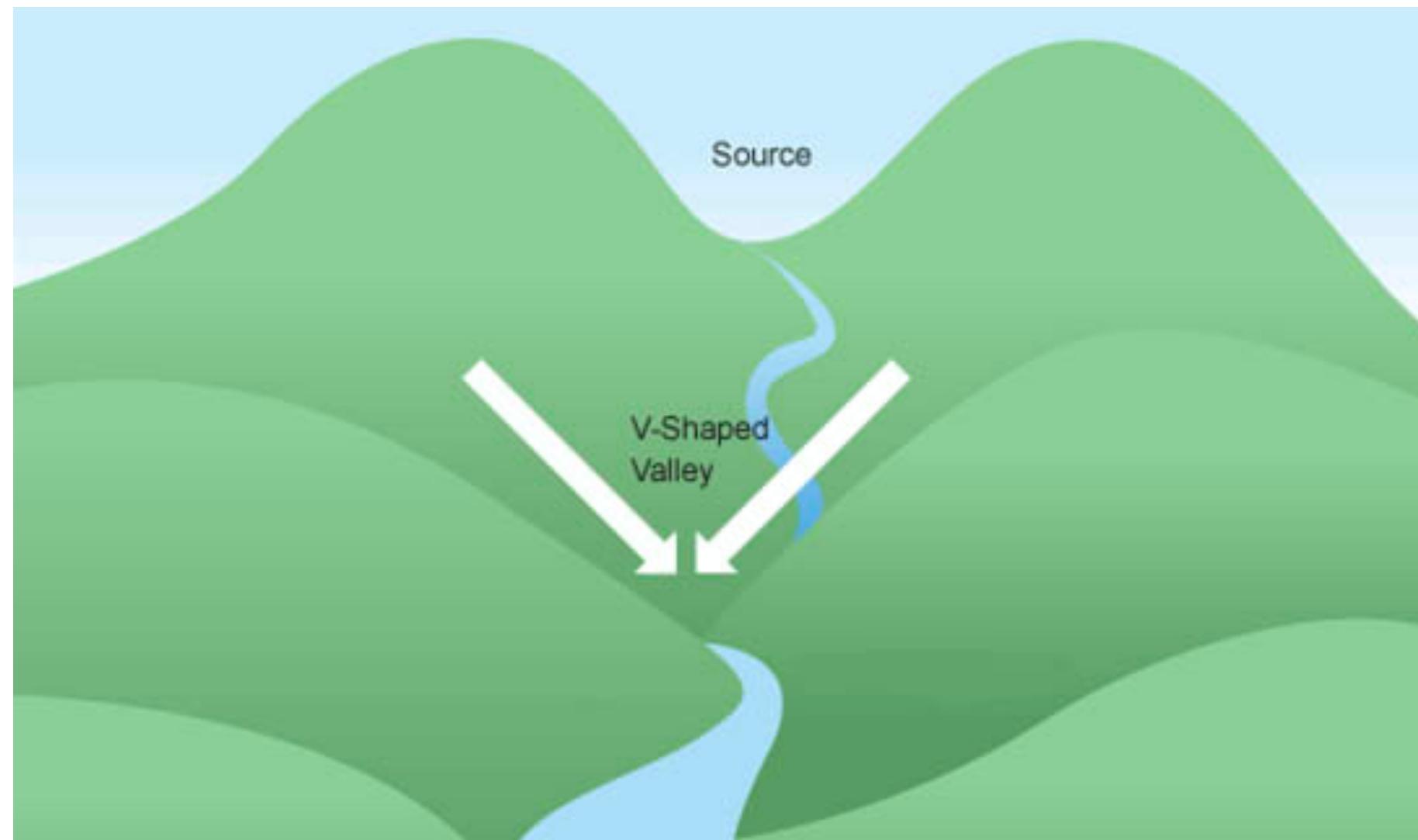
- **Optimization** $\mathbf{W}^* := \arg \min L(W) = \arg \min L_i(W)/N$

Error Landscape

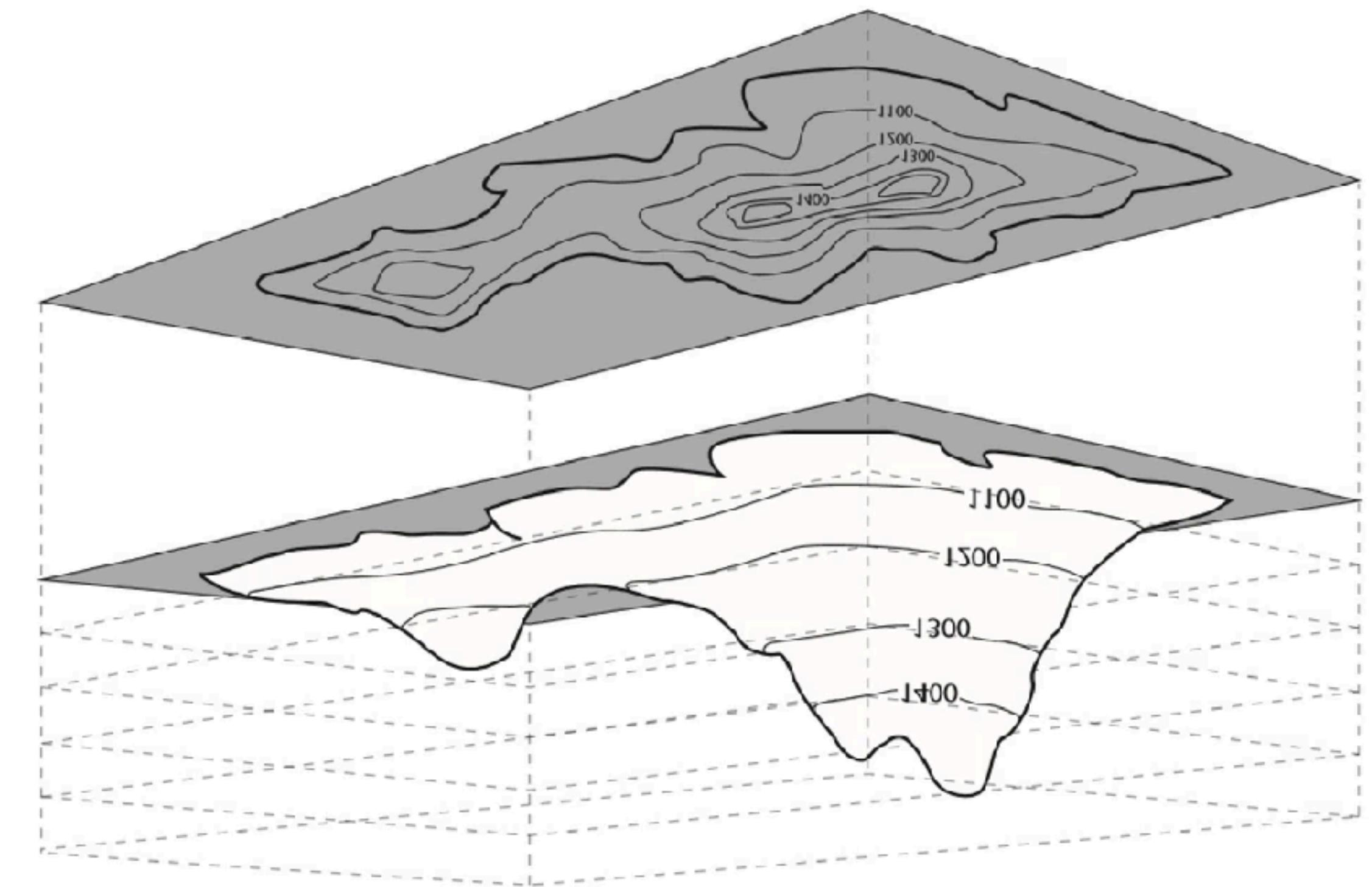
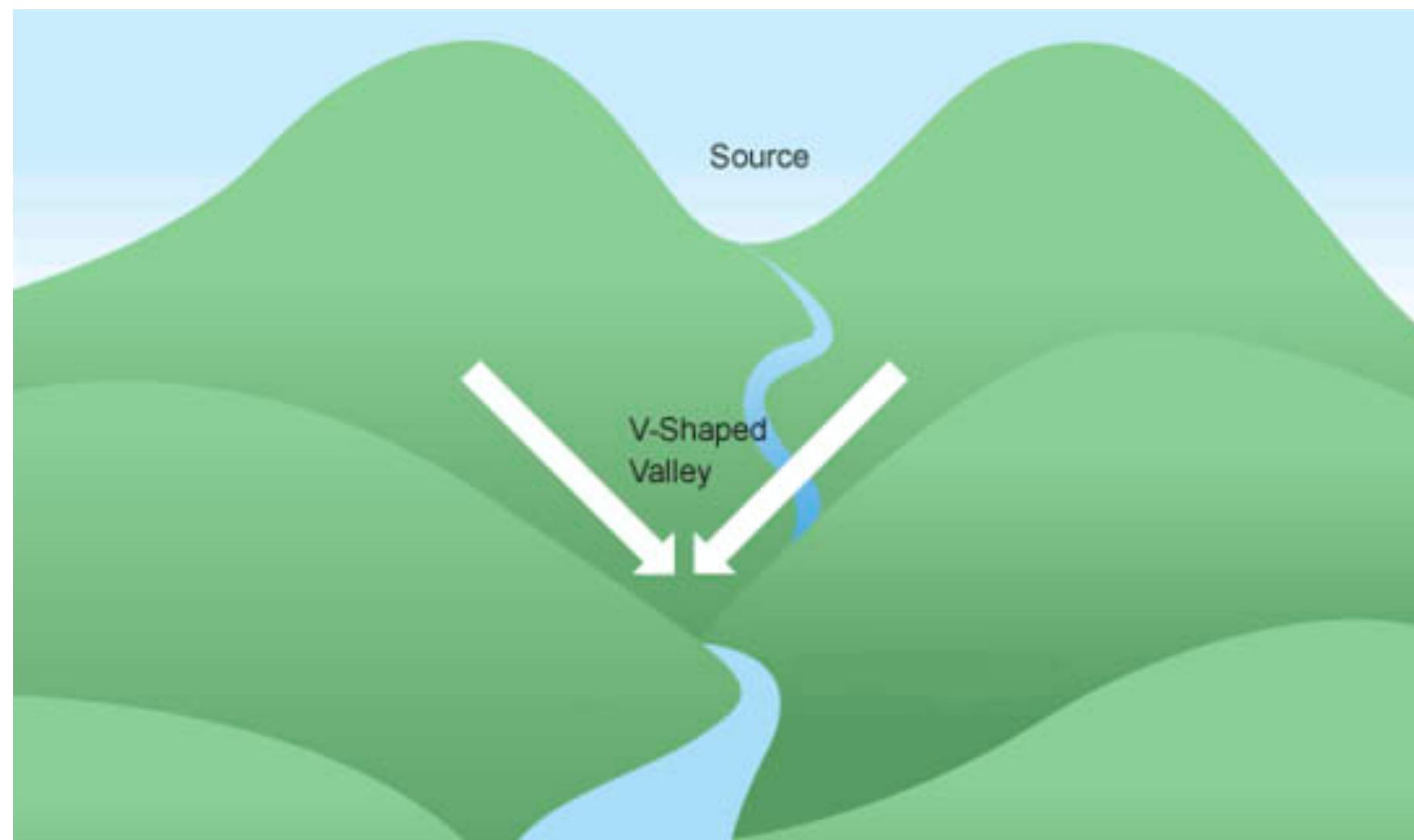
Error Landscape



Error Landscape



Error Landscape



Linear versus Nonlinear Models

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

Linear versus Nonlinear Models

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

- **Objective function #1**

$$L_i(W) := (W\mathbf{x}_i - y_i)^2$$

Linear versus Nonlinear Models

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

- **Objective function #1** $L_i(W) := (Wx_i - y_i)^2$

- **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

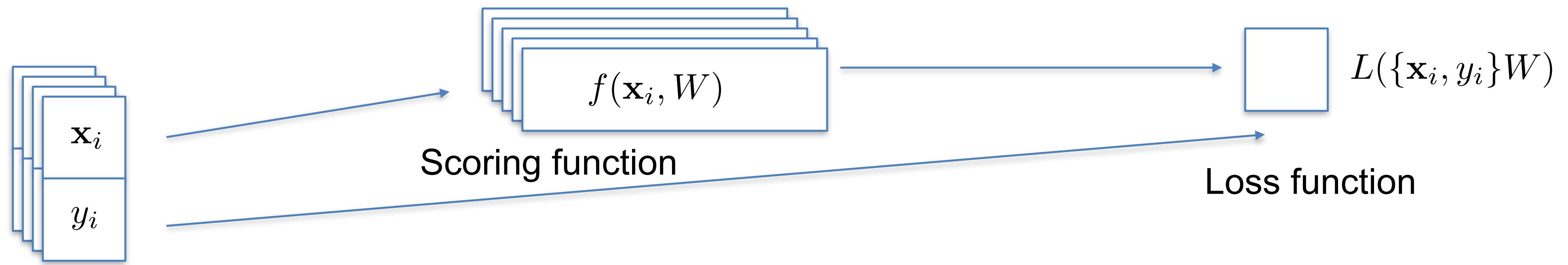
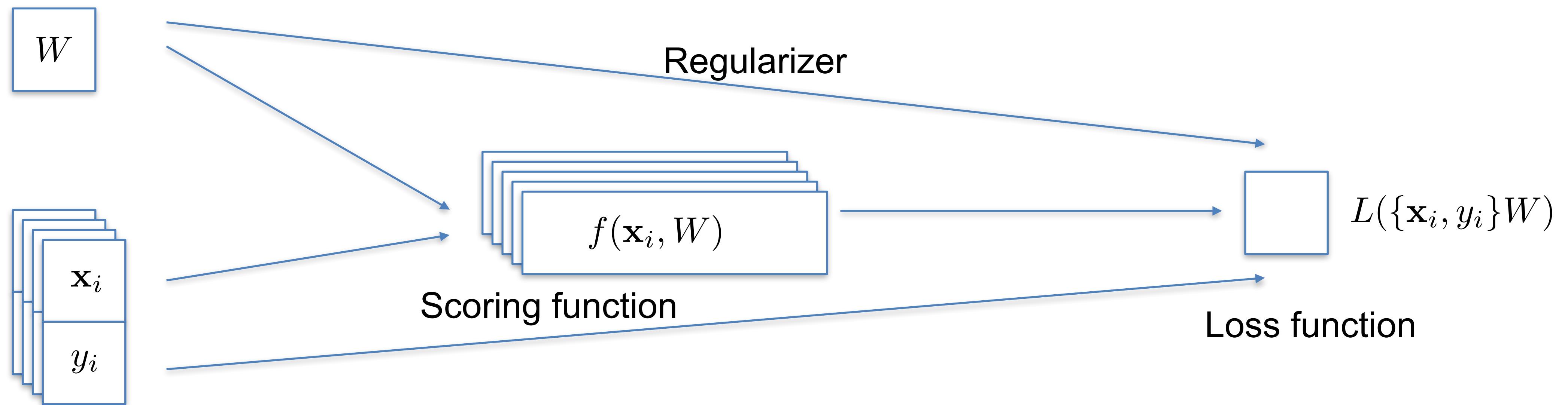


Image Classifier



Method 1a: Random search

- `bestLoss = infinity`

- `Loop`

 - `Pick random W`

 - `Compute loss`

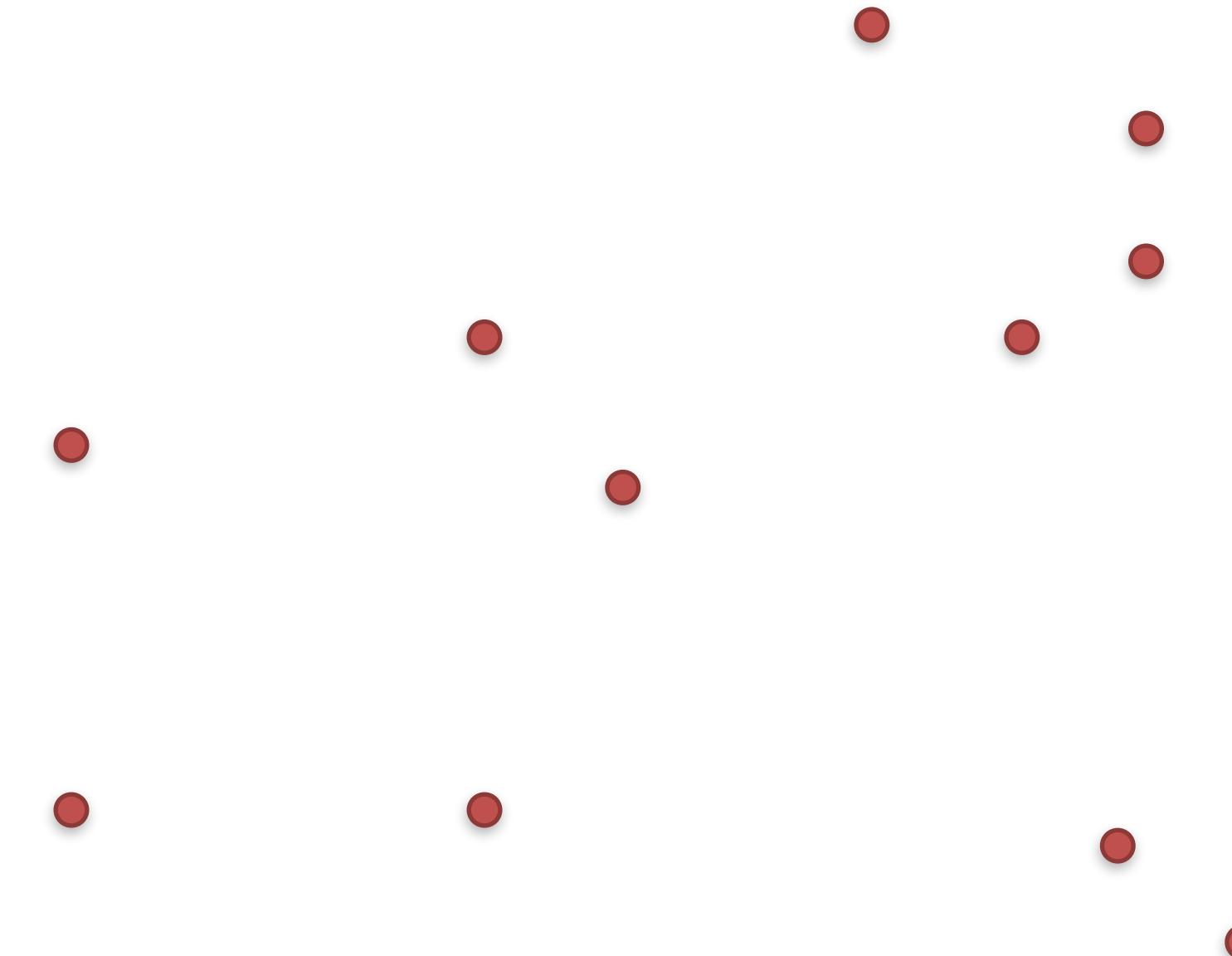
 - `if loss < bestLoss`

 - `bestLoss = loss`

 - `currW = W`

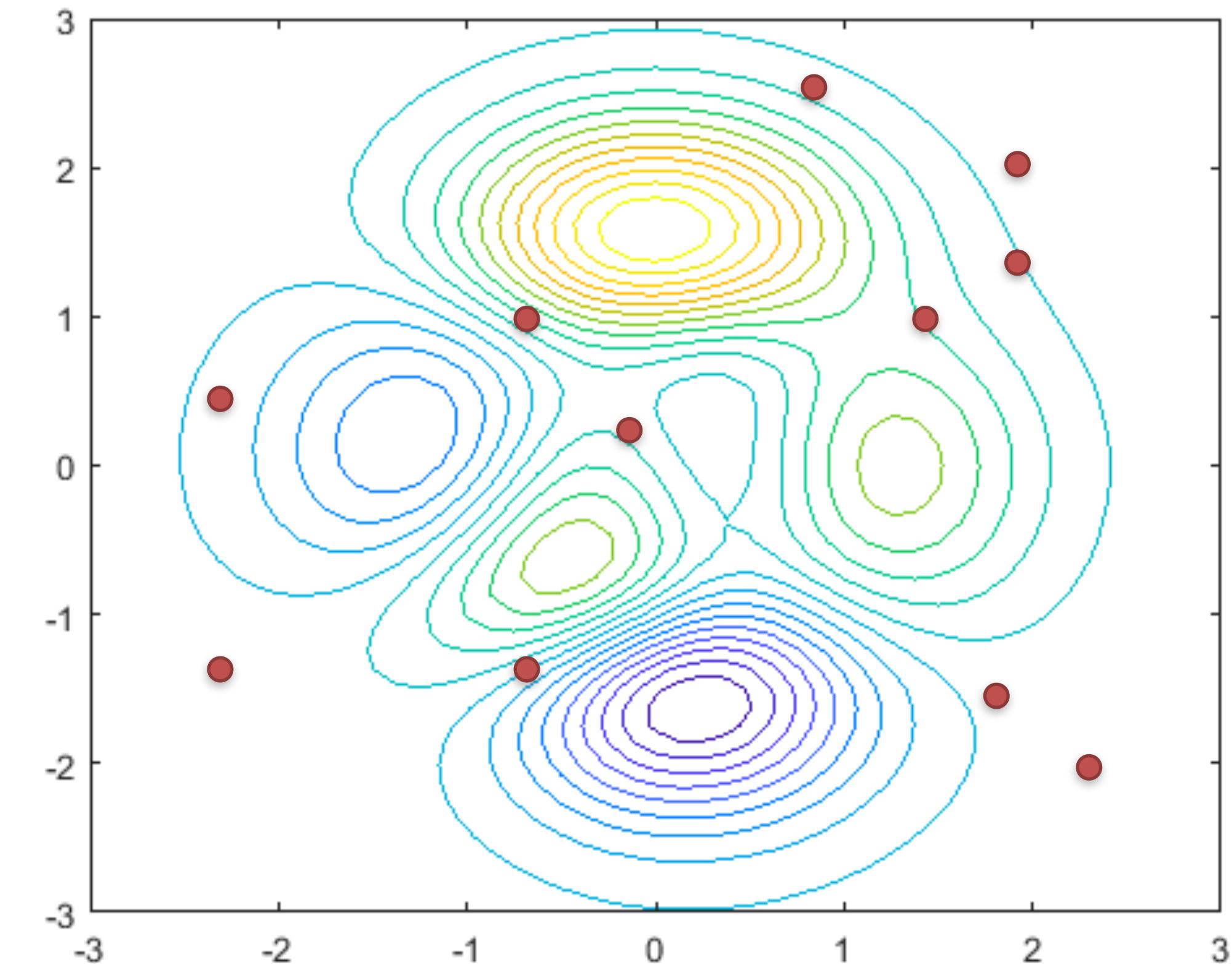
 - `endif`

- `endLoop`



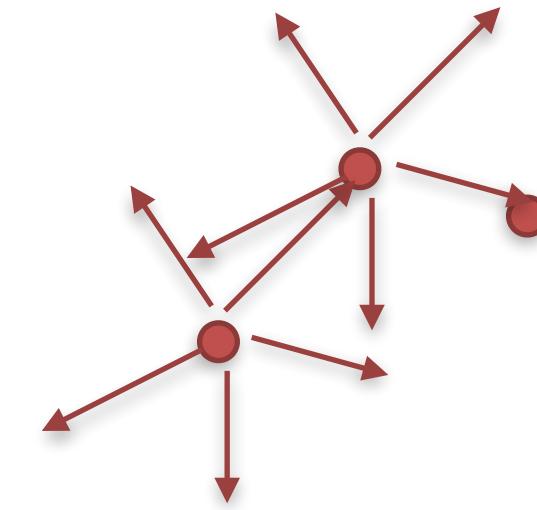
Method 1a: Random search

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



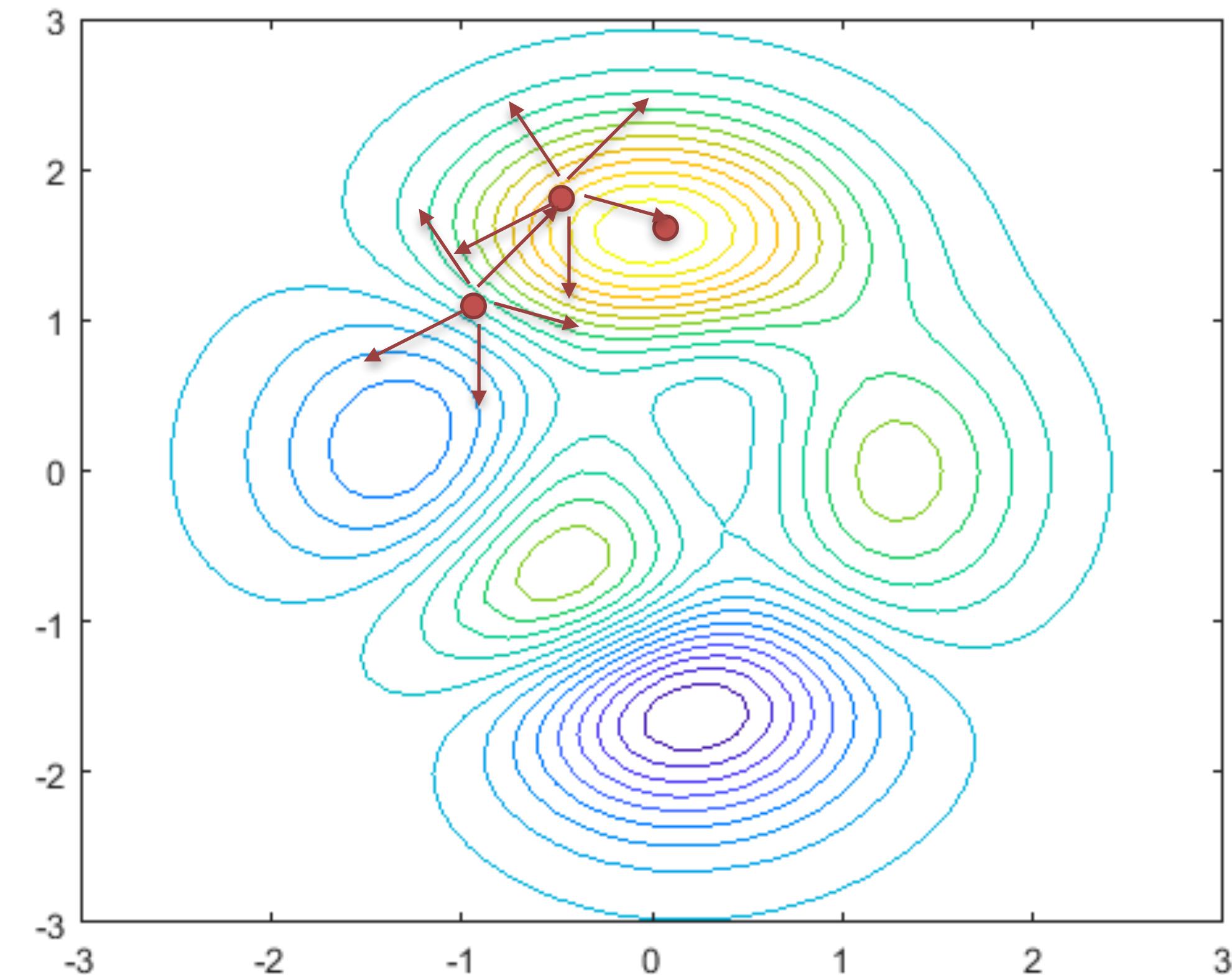
Method 1b: Random search++

- `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**



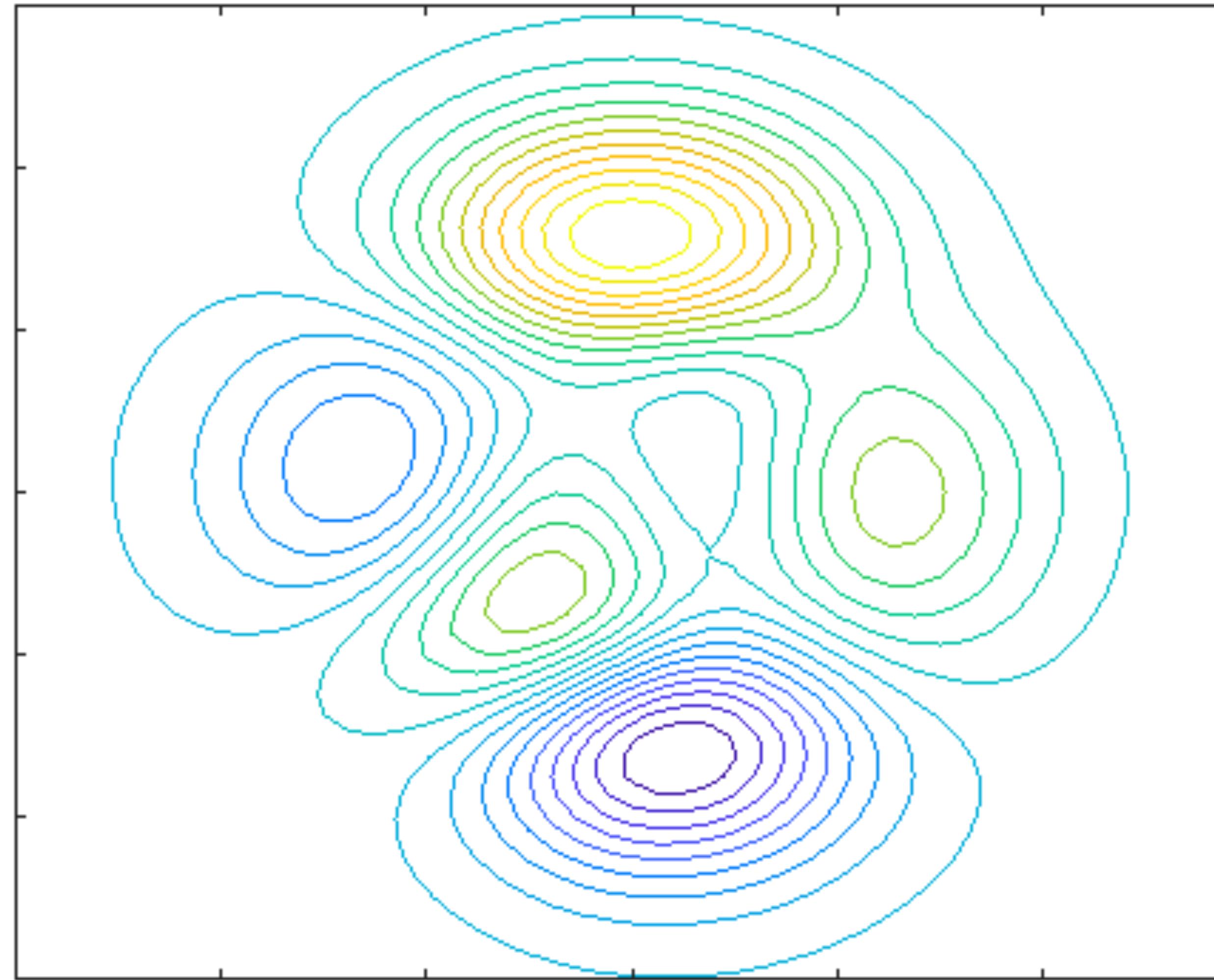
Optimize

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

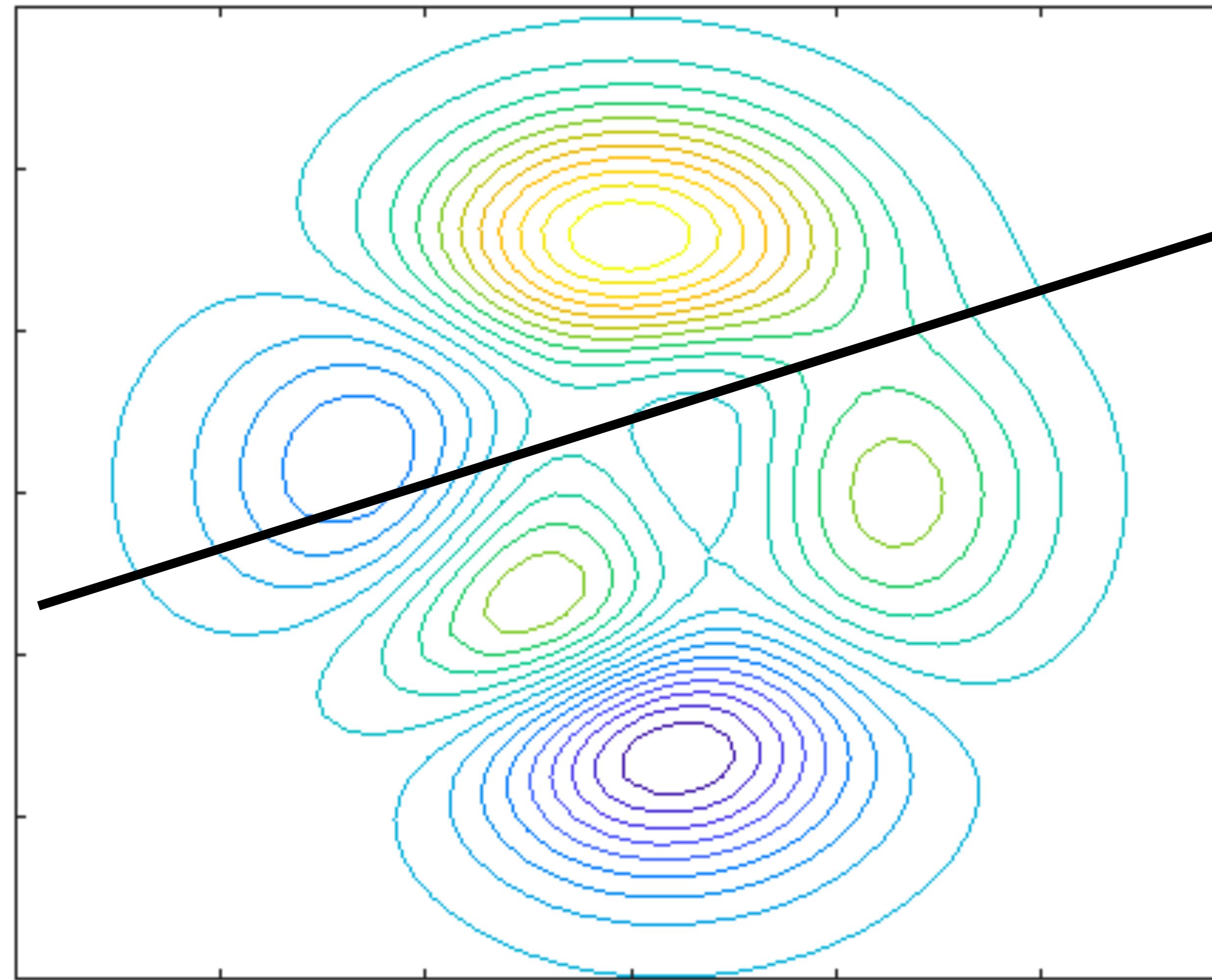
$$\nabla g(\mathbf{x}_i, W) = \left[\frac{\partial g(\mathbf{x}_i, W)}{\partial w_i} = 0 \right]_{D \times 1}$$

Method 2: Follow Negative of Gradient

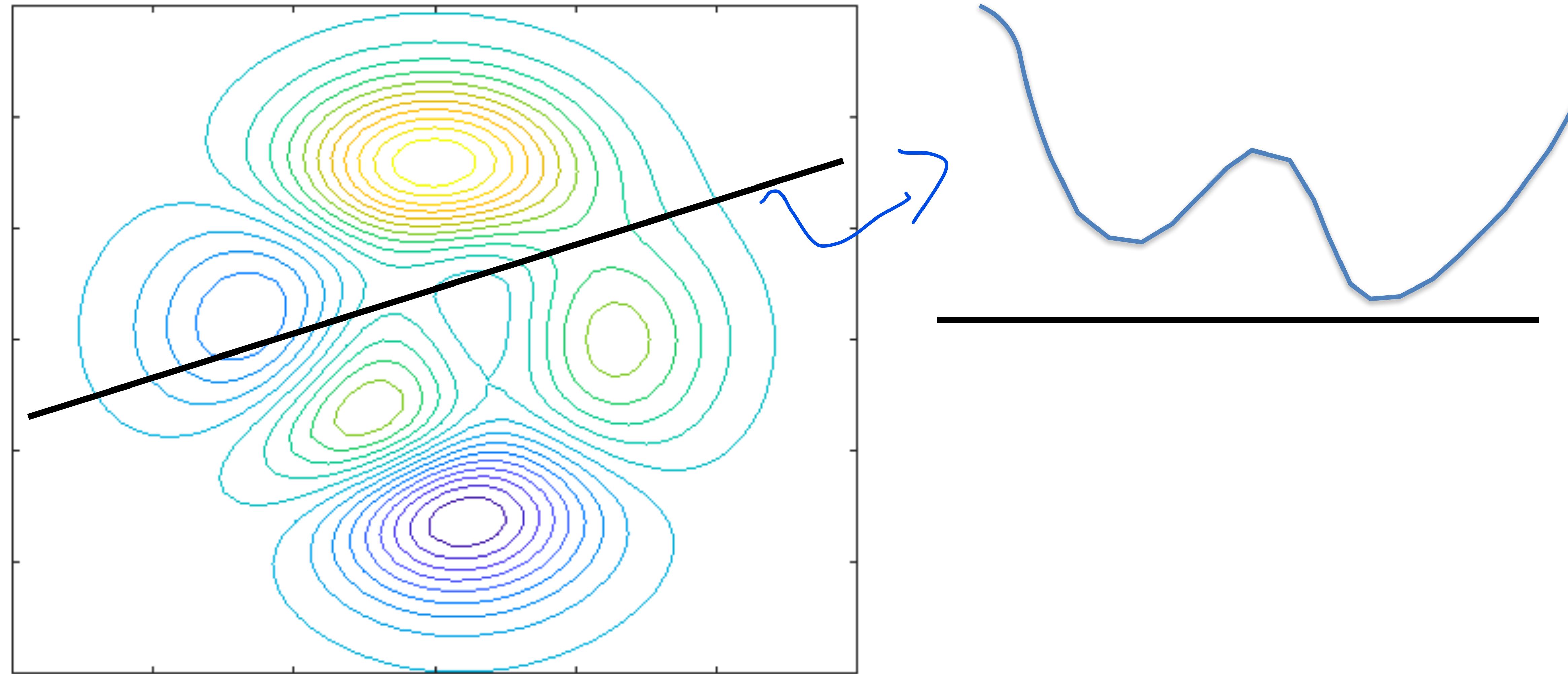
Method 2: Follow Negative of Gradient



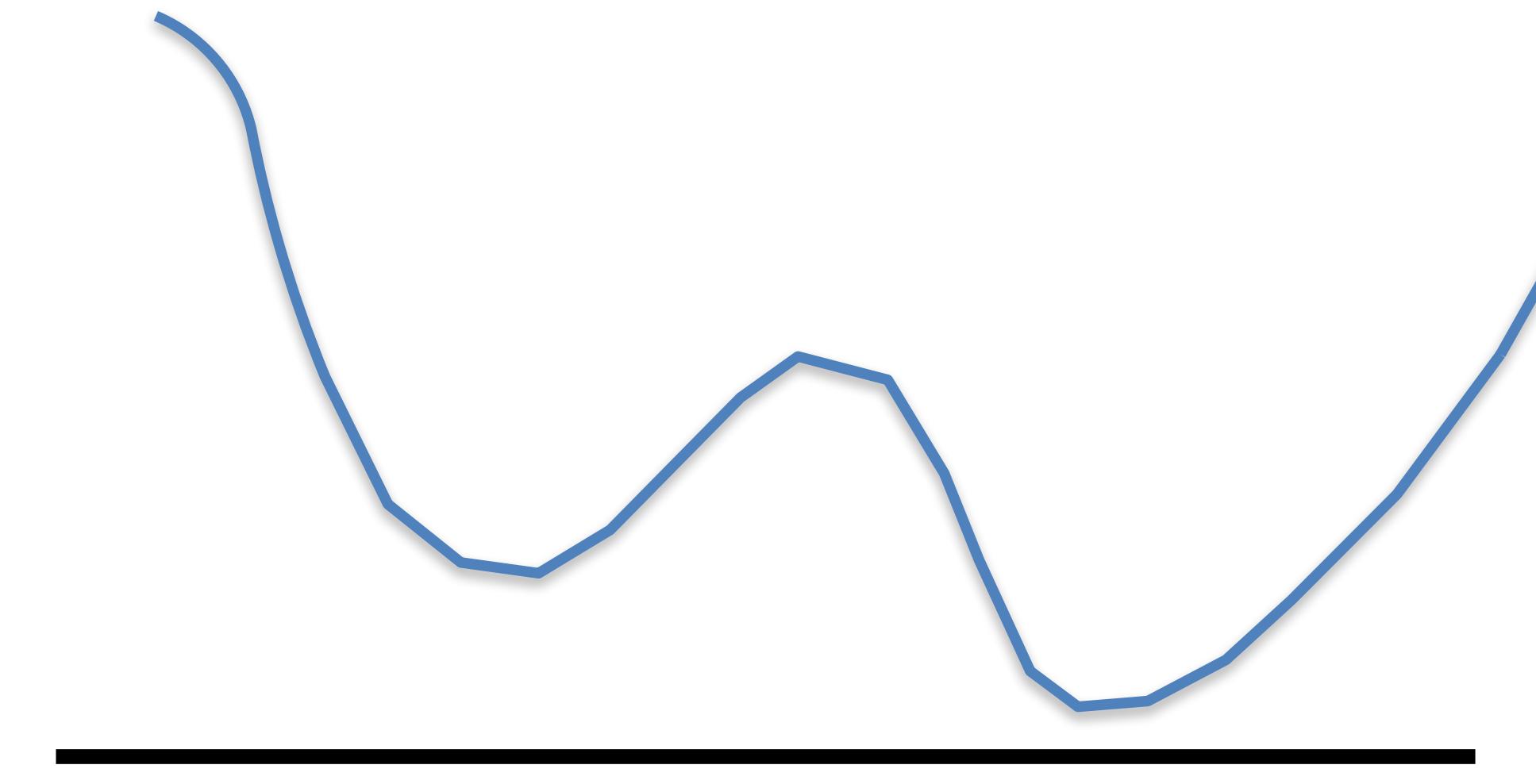
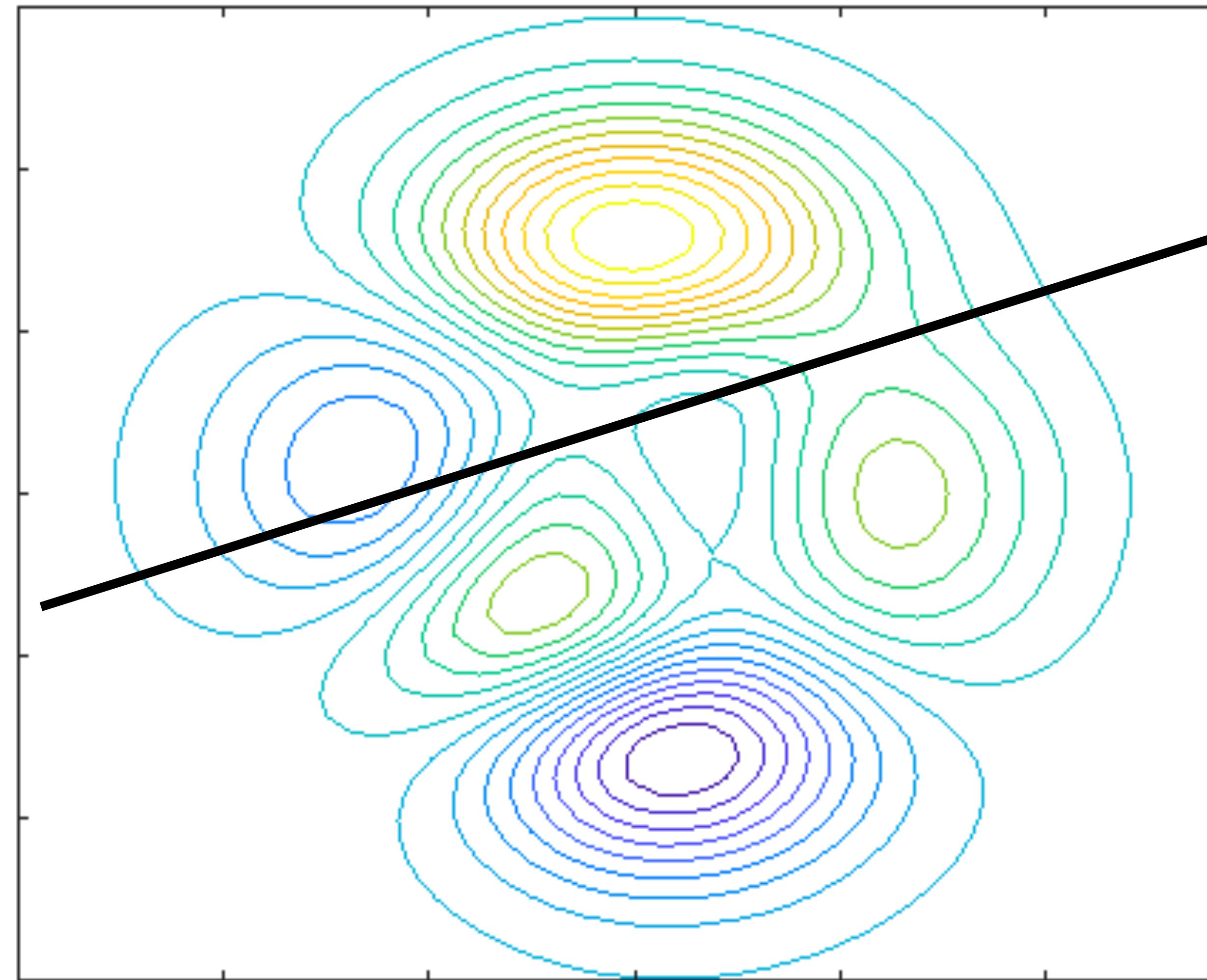
Method 2: Follow Negative of Gradient



Method 2: Follow Negative of Gradient

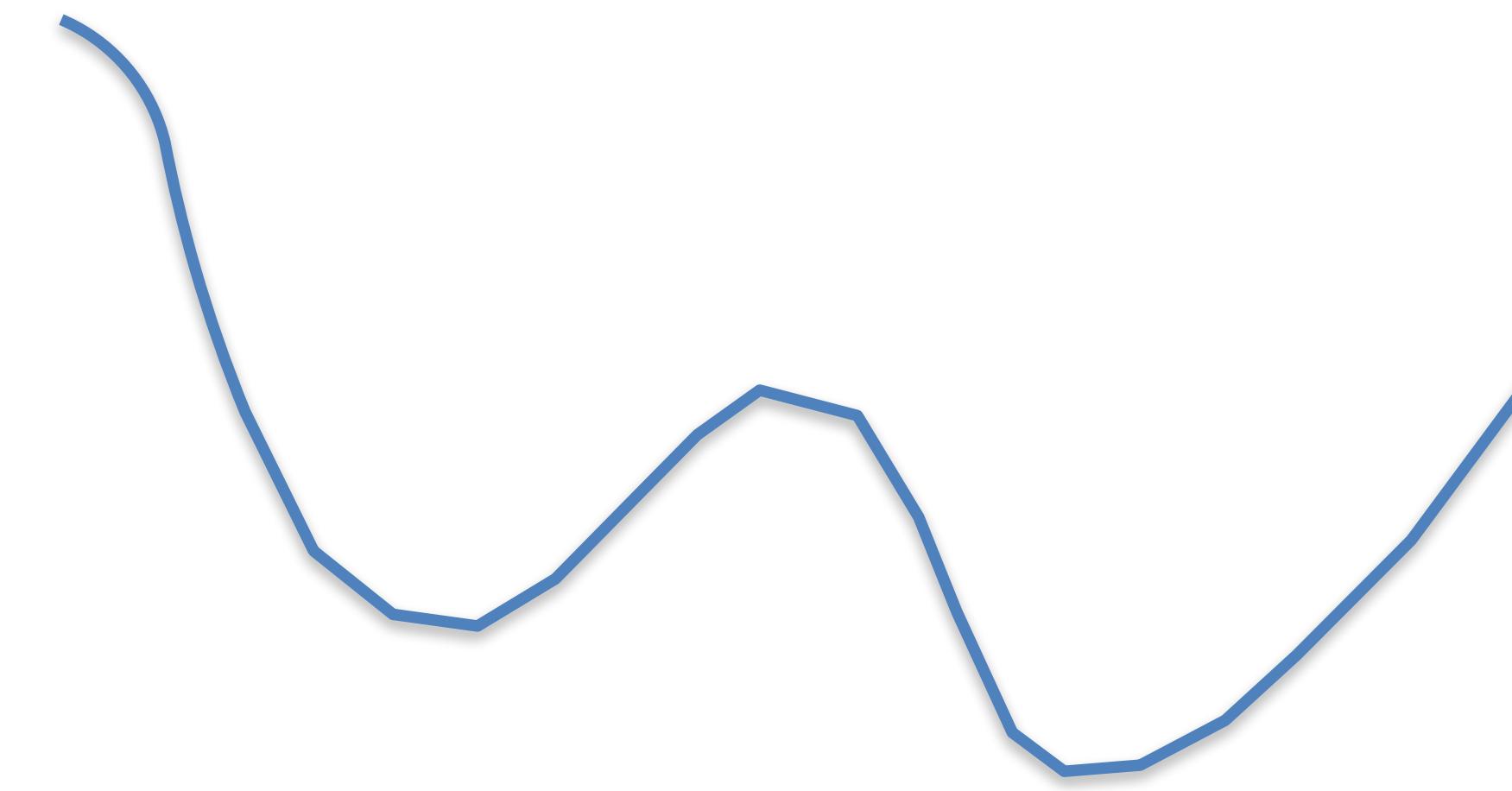


Method 2: Follow Negative of Gradient

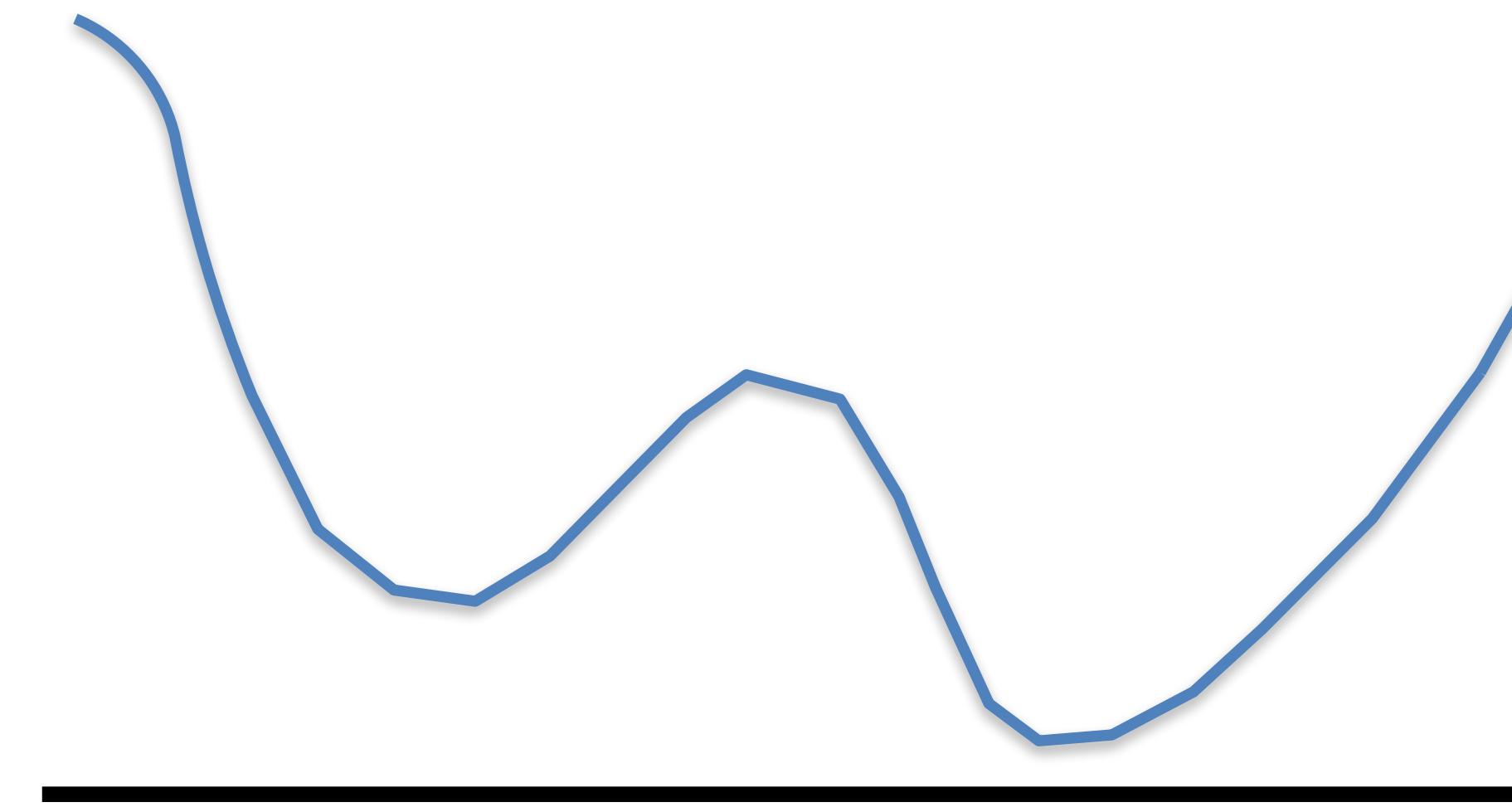


$$\frac{df(x)}{dx}$$

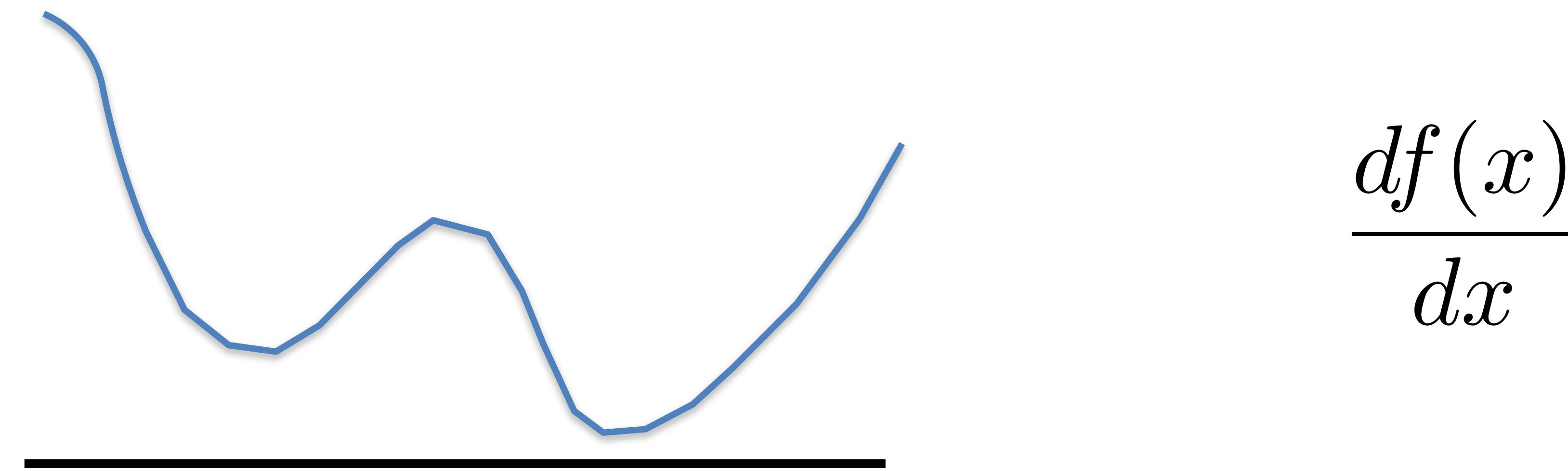
Method 2: Follow Negative of Gradient



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**

Perceptron Algorithm [Rosenblatt '57]

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\mathbf{x}_i^T \mathbf{w}, y_i)$$

Perceptron Algorithm [Rosenblatt '57]

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\mathbf{x}_i^T \mathbf{w}, y_i)$$

$$\mathcal{L}(z, y_i) = \begin{cases} 0 & \text{if } y_i z \geq 0 \\ -y_i z & \text{else.} \end{cases}$$

Perceptron Algorithm [Rosenblatt '57]

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\mathbf{x}_i^T \mathbf{w}, y_i)$$

$$= \frac{1}{n} \sum_{i \in V} -y_i (\mathbf{x}_i^T \mathbf{w}) \quad \text{where } V \text{ is the set of indices with } y_i(\mathbf{x}_i^T \mathbf{w}) < 0$$

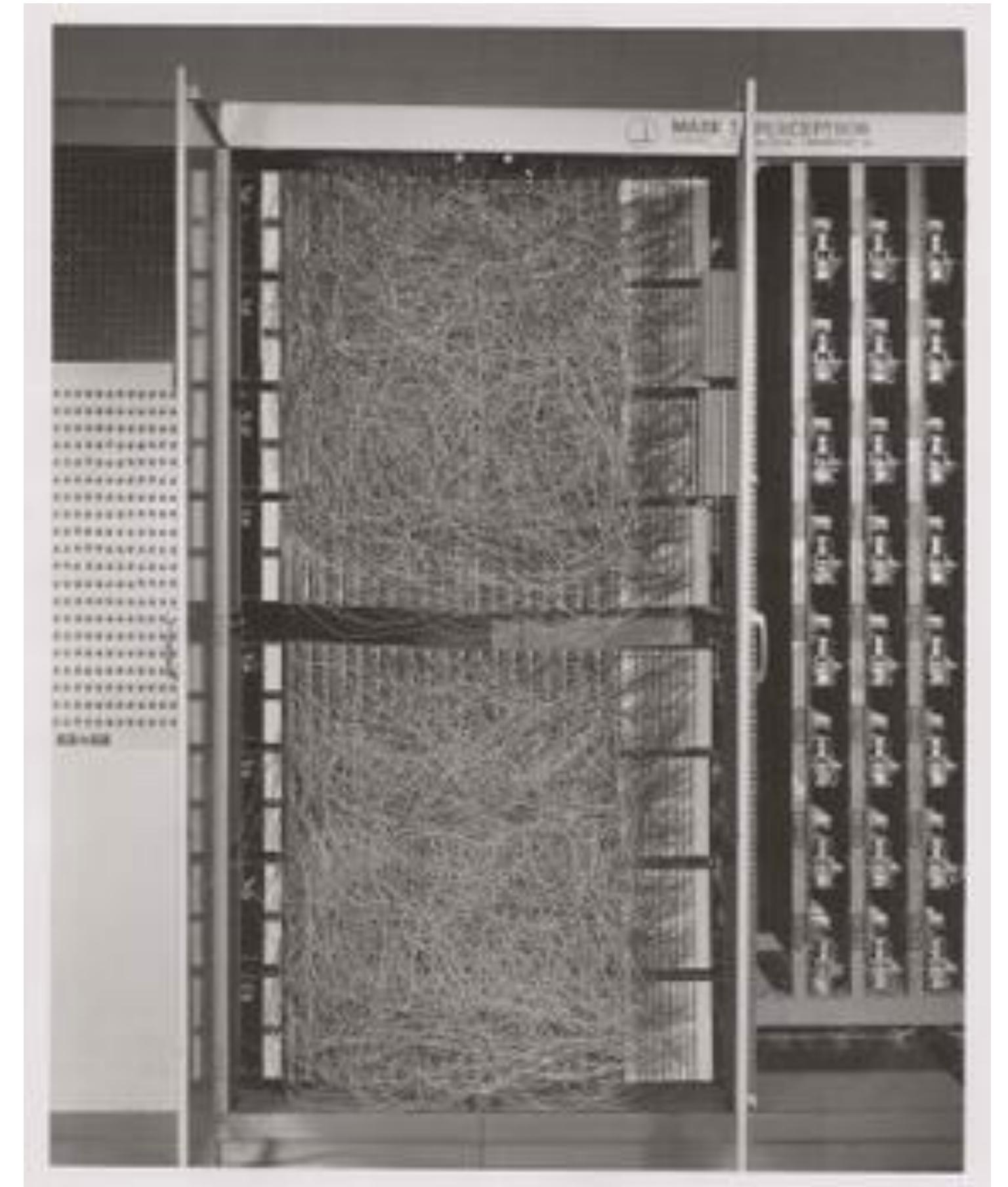
$$\mathcal{L}(z, y_i) = \begin{cases} 0 & \text{if } y_i z \geq 0 \\ -y_i z & \text{else.} \end{cases}$$

Perceptron Algorithm [Rosenblatt '57]

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\mathbf{x}_i^T \mathbf{w}, y_i)$$

$$= \frac{1}{n} \sum_{i \in V} -y_i (\mathbf{x}_i^T \mathbf{w}) \quad \text{where } V \text{ is the set of indices with } y_i(\mathbf{x}_i^T \mathbf{w}) < 0$$

$$\mathcal{L}(z, y_i) = \begin{cases} 0 & \text{if } y_i z \geq 0 \\ -y_i z & \text{else.} \end{cases}$$



<https://en.wikipedia.org/wiki/Perceptron>

Perceptron Algorithm [Rosenblatt '57]

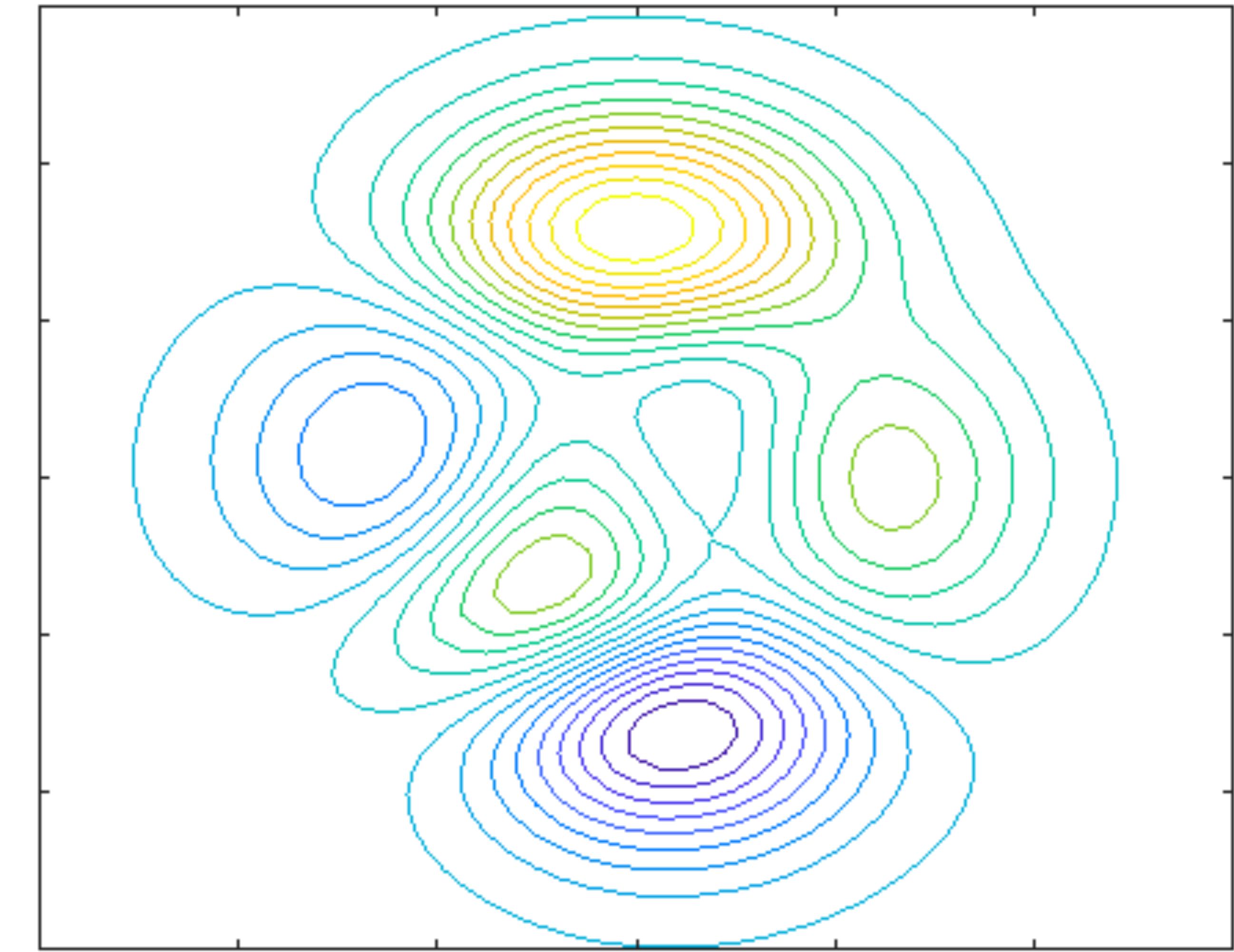
- Initialize \mathbf{w}
- While $\mathcal{L}(\mathbf{w}) > 0$
 - $V \leftarrow \text{set of indices with } y_i(\mathbf{x}_i^T \mathbf{w}) < 0$
 - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \sum_{i \in V} -y_i \mathbf{x}_i$

Gradient Descent

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

Gradient Descent

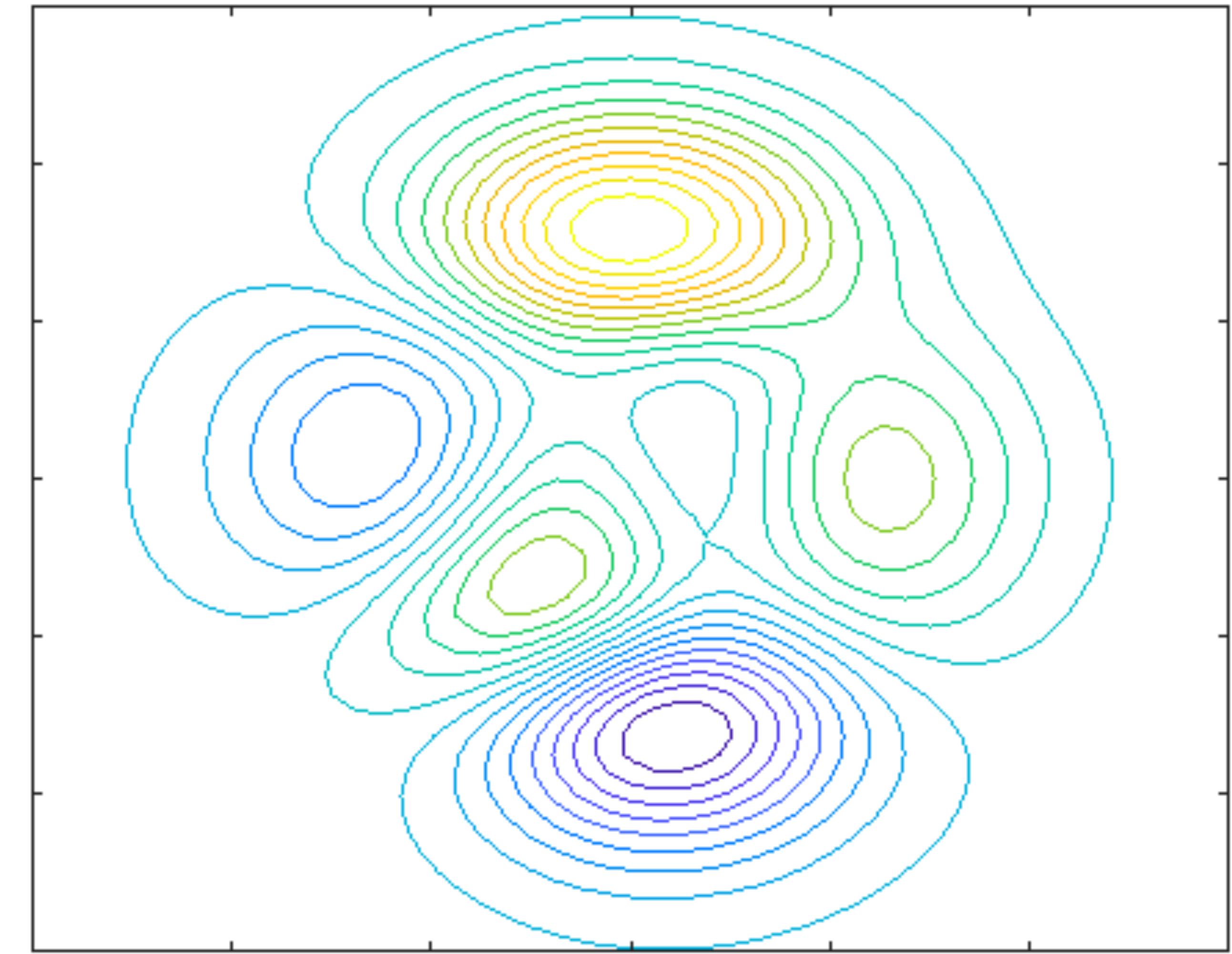
$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$



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
```



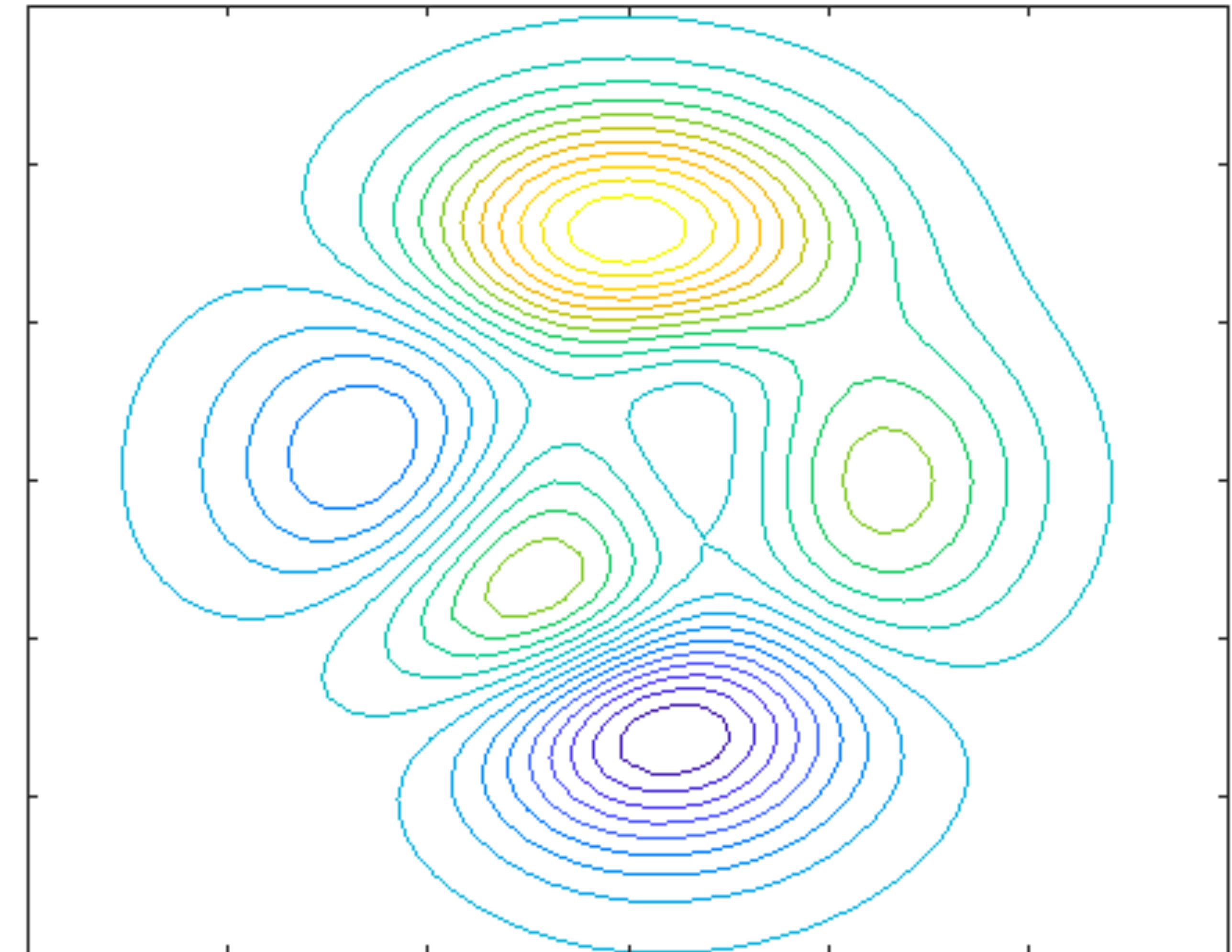
Gradient Descent

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

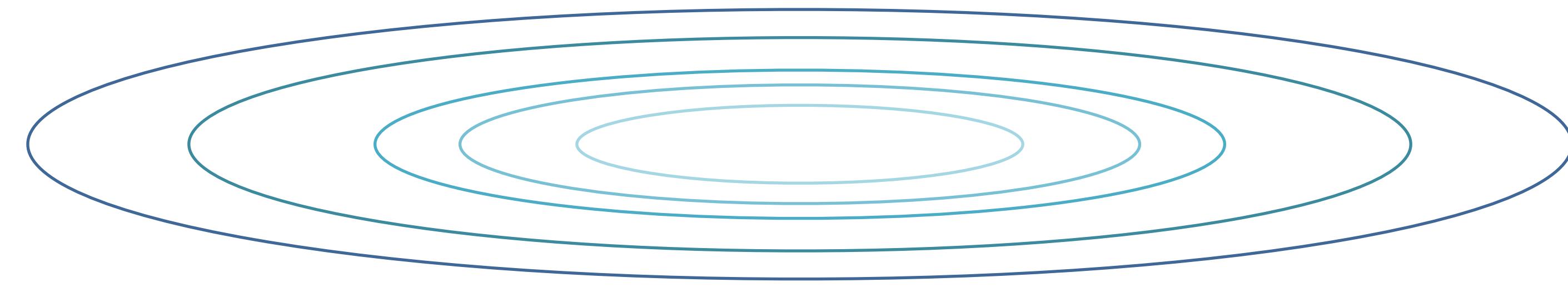
```
w = initialize_weights()
for t in range(num_steps):
    dw = compute_gradient(loss_fn, data, w)
    w -= learning_rate * dw
```

Hyperparameters?

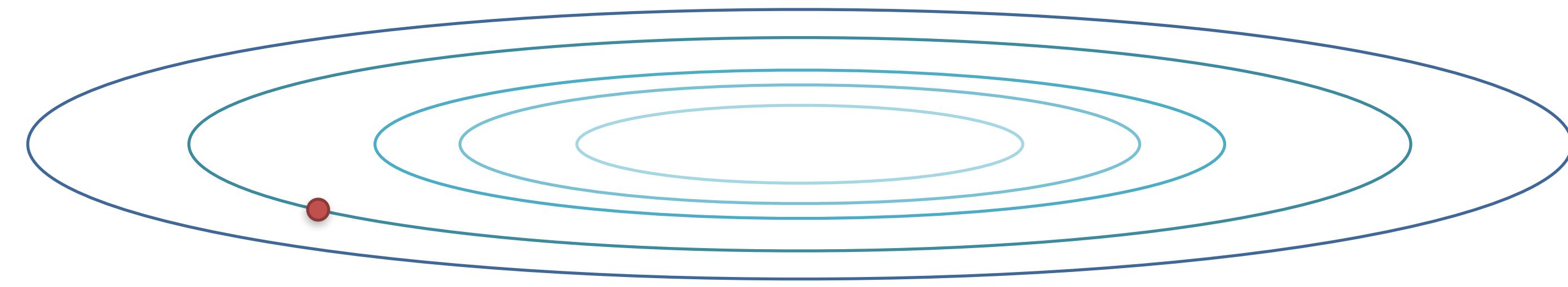
- a) intialize_weights
- b) learning_rate
- c) number of steps/truncation condition



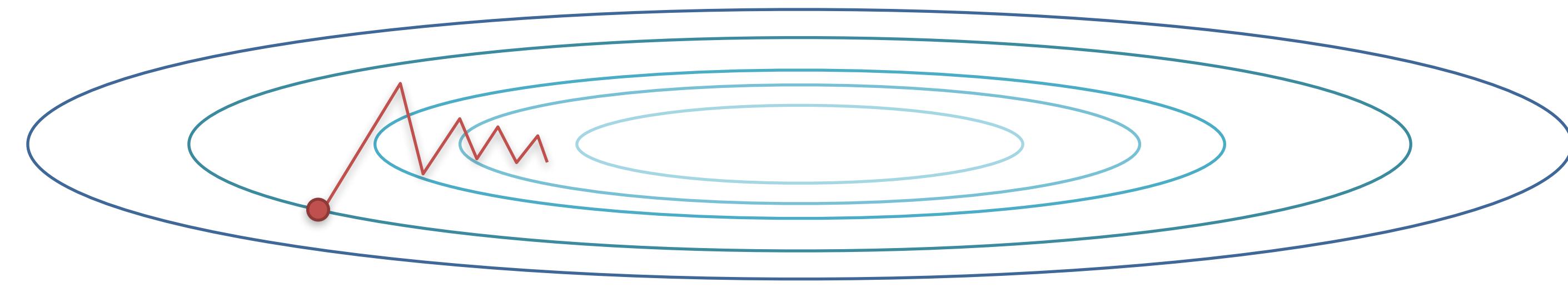
Problems with Gradient Descent



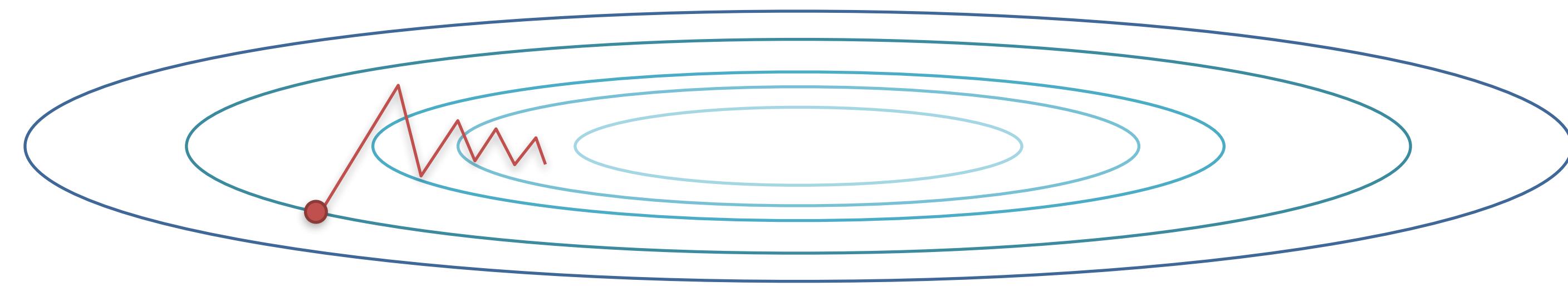
Problems with Gradient Descent



Problems with Gradient Descent

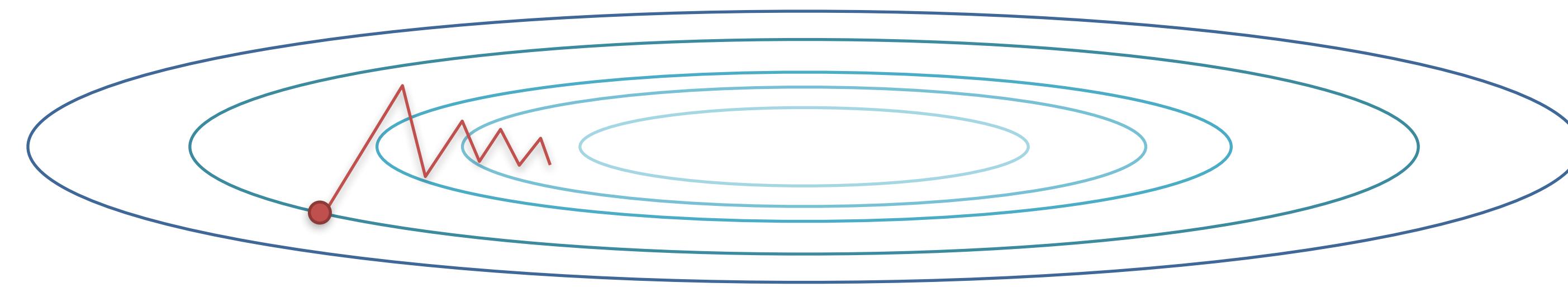


Problems with Gradient Descent



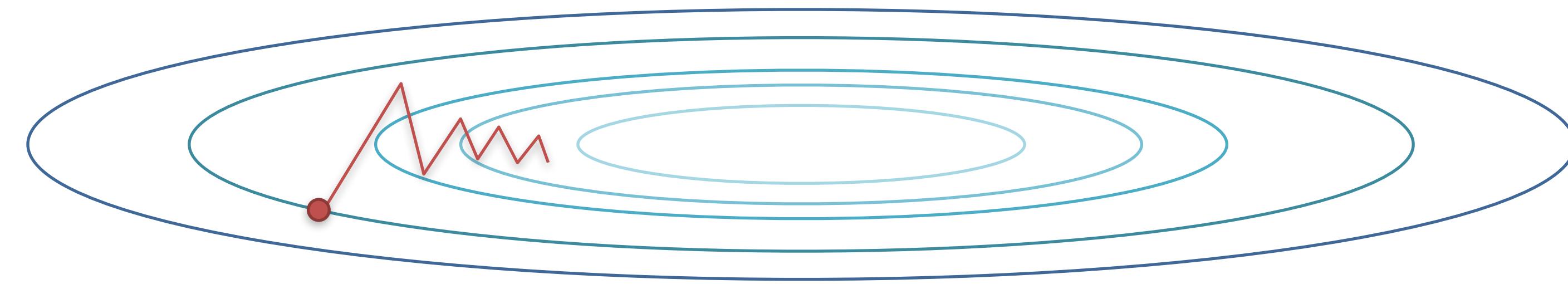
1. Bad condition number → slow convergence

Problems with Gradient Descent



1. Bad condition number → slow convergence
2. Local minima

Problems with Gradient Descent



1. Bad condition number → slow convergence
2. Local minima
3. Saddle point

Stochastic/Batch Gradient Descent

Stochastic/Batch 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/Batch 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/Batch Gradient Descent

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

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

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

Stochastic/Batch Gradient Descent

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

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

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

$$m \ll N$$

Stochastic/Batch Gradient Descent

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

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

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

Hyperparameters?

- a) initialize_weights
- b) learning_rate
- c) number of steps/truncation condition
- d) batch size (8, 16, 32, ...)
- e) sampling

$$m \ll N$$

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

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

Linear Classification Loss Visualization

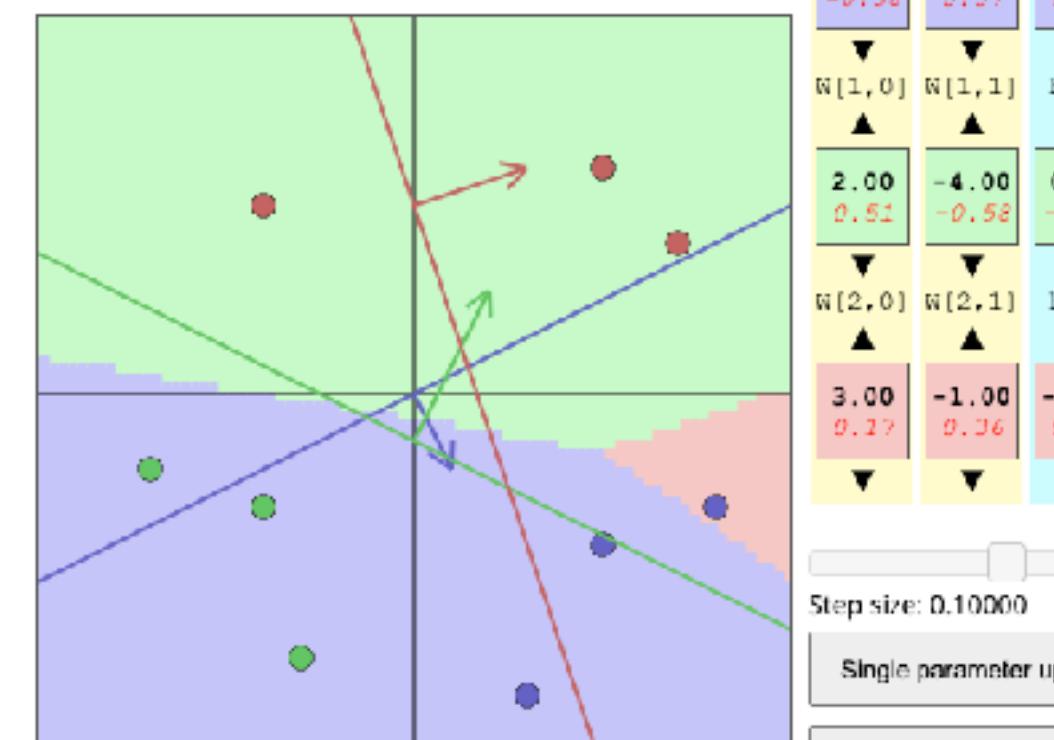
These linear classifiers were written in Javascript for Stanford's CS231n: Convolutional Neural Networks for Visual Recognition.

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 $[3 \times 2]$ and the bias vector is of size $[3 \times 1]$. 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 $[3 \times 1]$ vector that holds the class scores, the loss has the form:

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

Where N is the number of examples, and λ is a hyperparameter that controls the strength of the L2 regularization penalty $R(W) = \sum_k \sum_l W_{kj}^2$. 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](#).

Data points 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 whichever class is most likely at any point according to the current weights. Each classifier is visualized by a line that with backprop) is in red. 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 parameters. 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	1.00	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

Step size: 0.10000

Single parameter update

Start repeated update

Stop repeated update

L2 Regularization strength: 0.10000

Multiclass SVM loss formulation:

Weston Watkins 1999

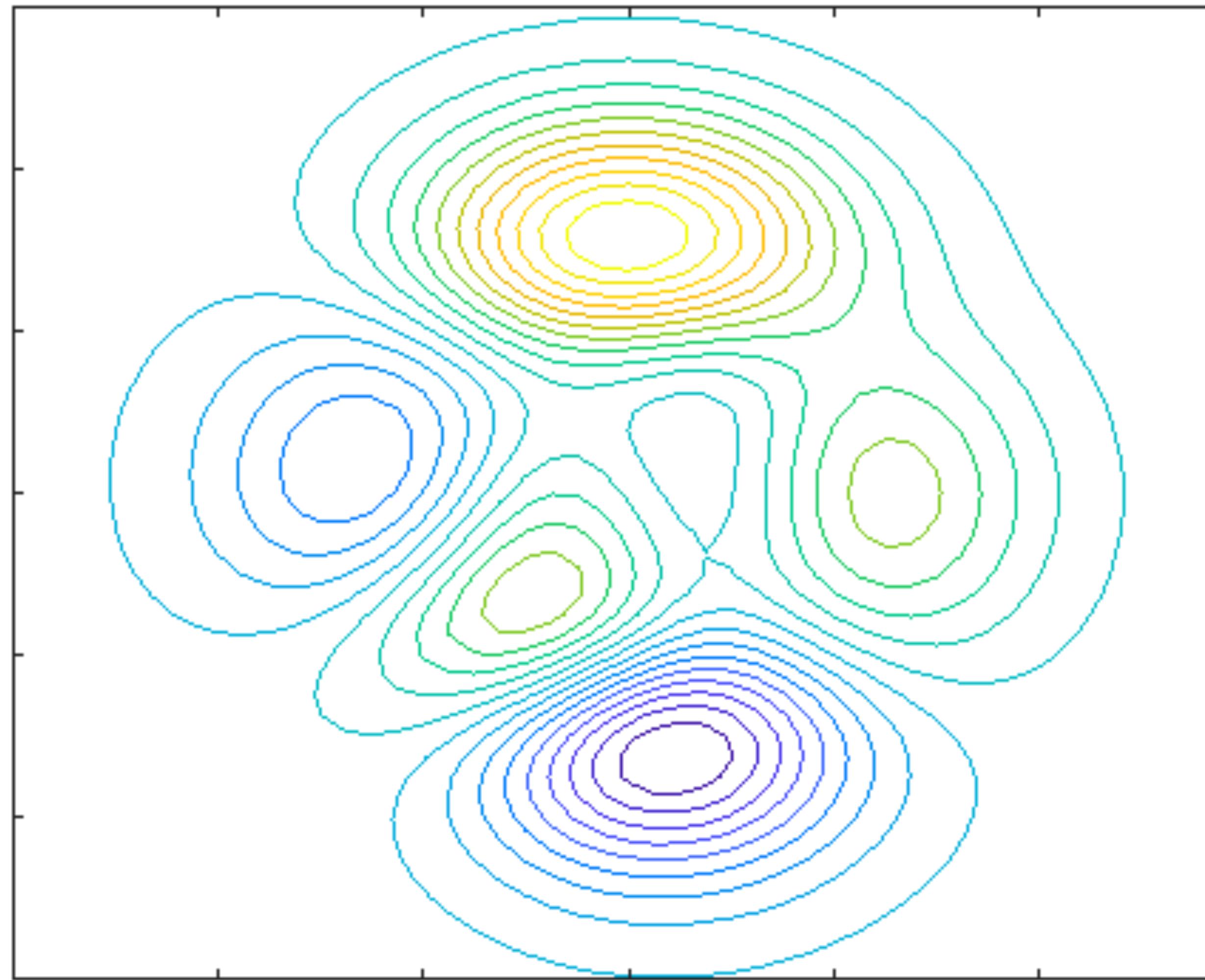
One vs. All

Structured SVM

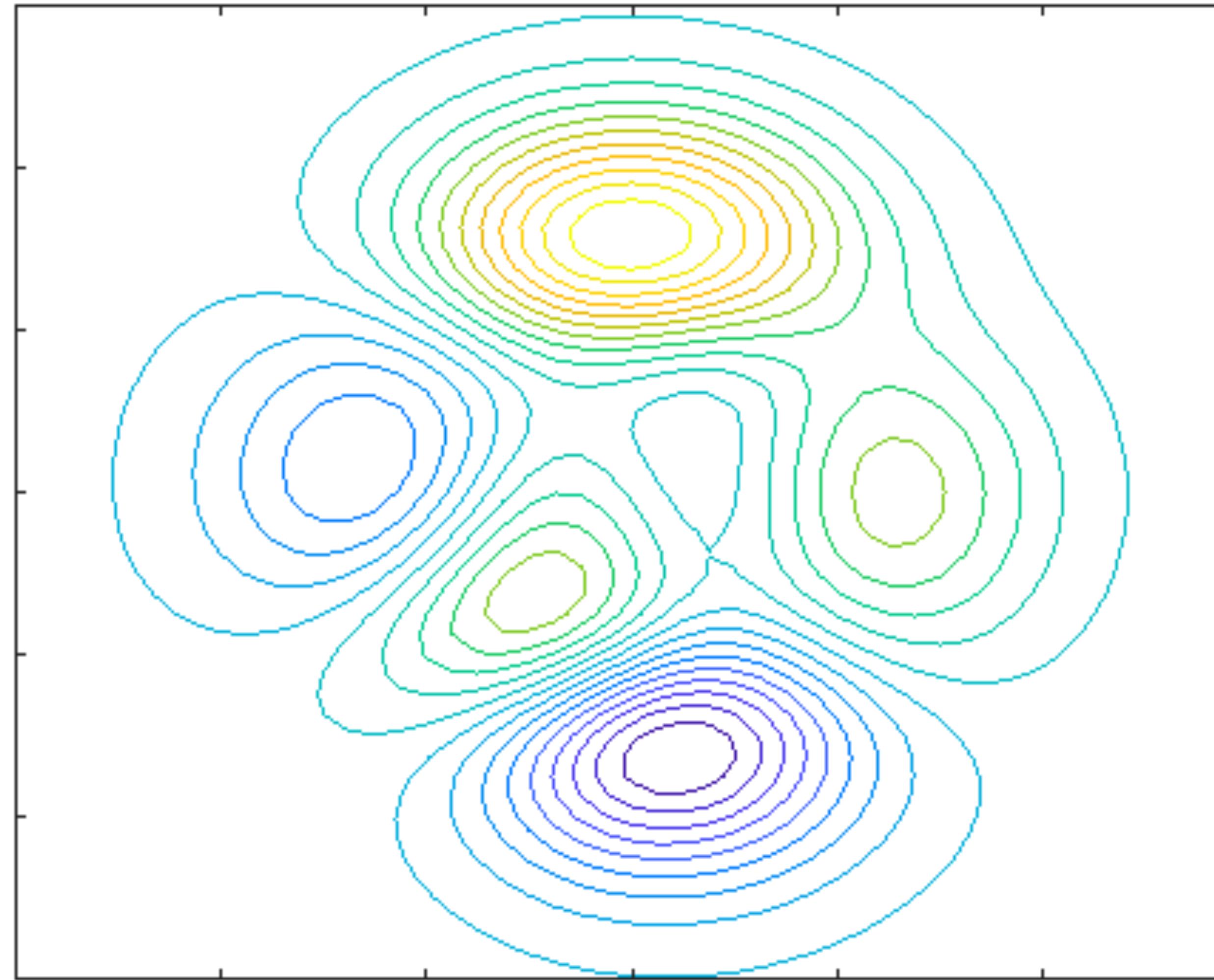
Softmax

Problems with GD (or SGD)

Problems with GD (or SGD)



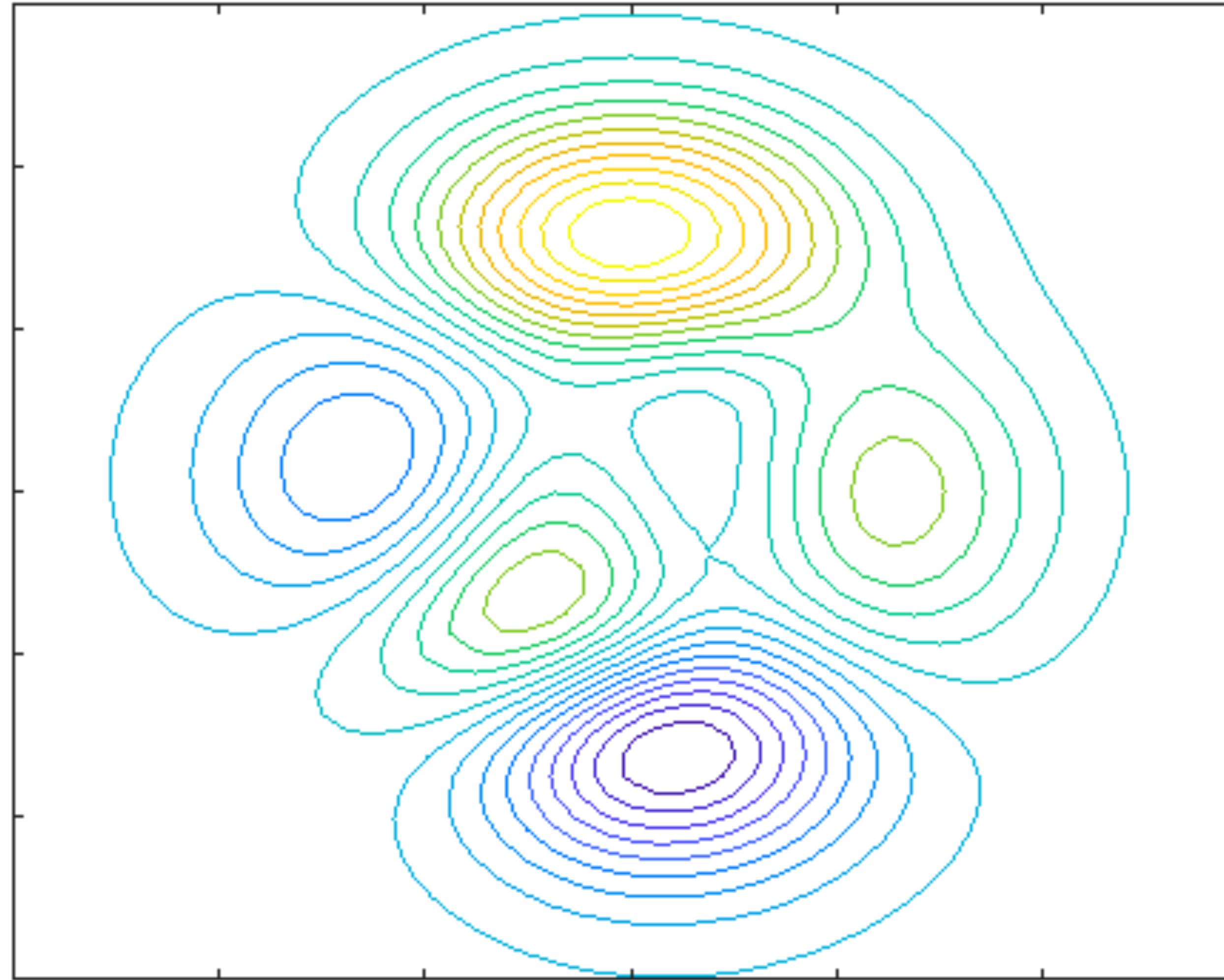
Problems with GD (or SGD)



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

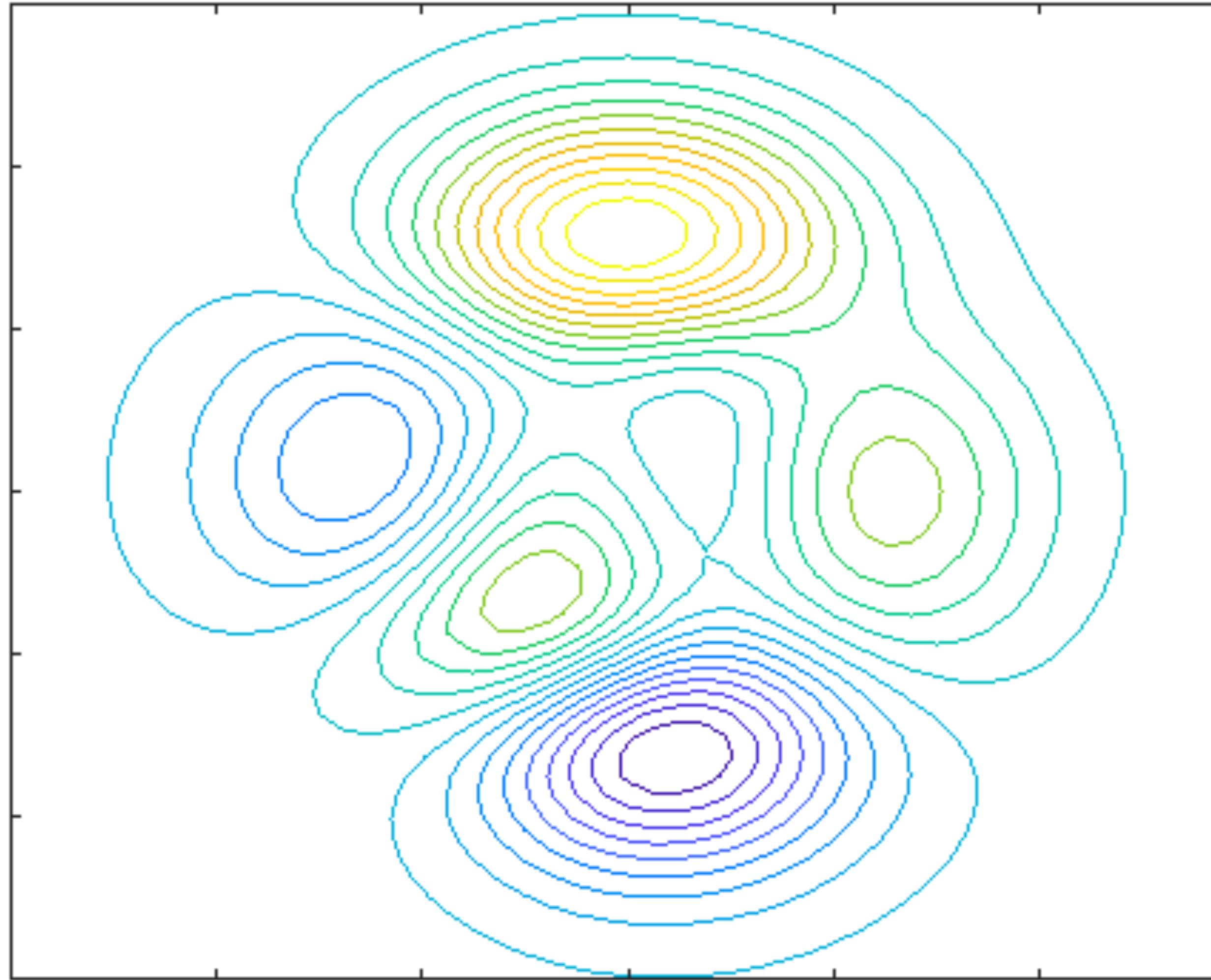
$$w_{t+1} = w_t - \alpha v_{t+1}$$

Problems with GD (or SGD)



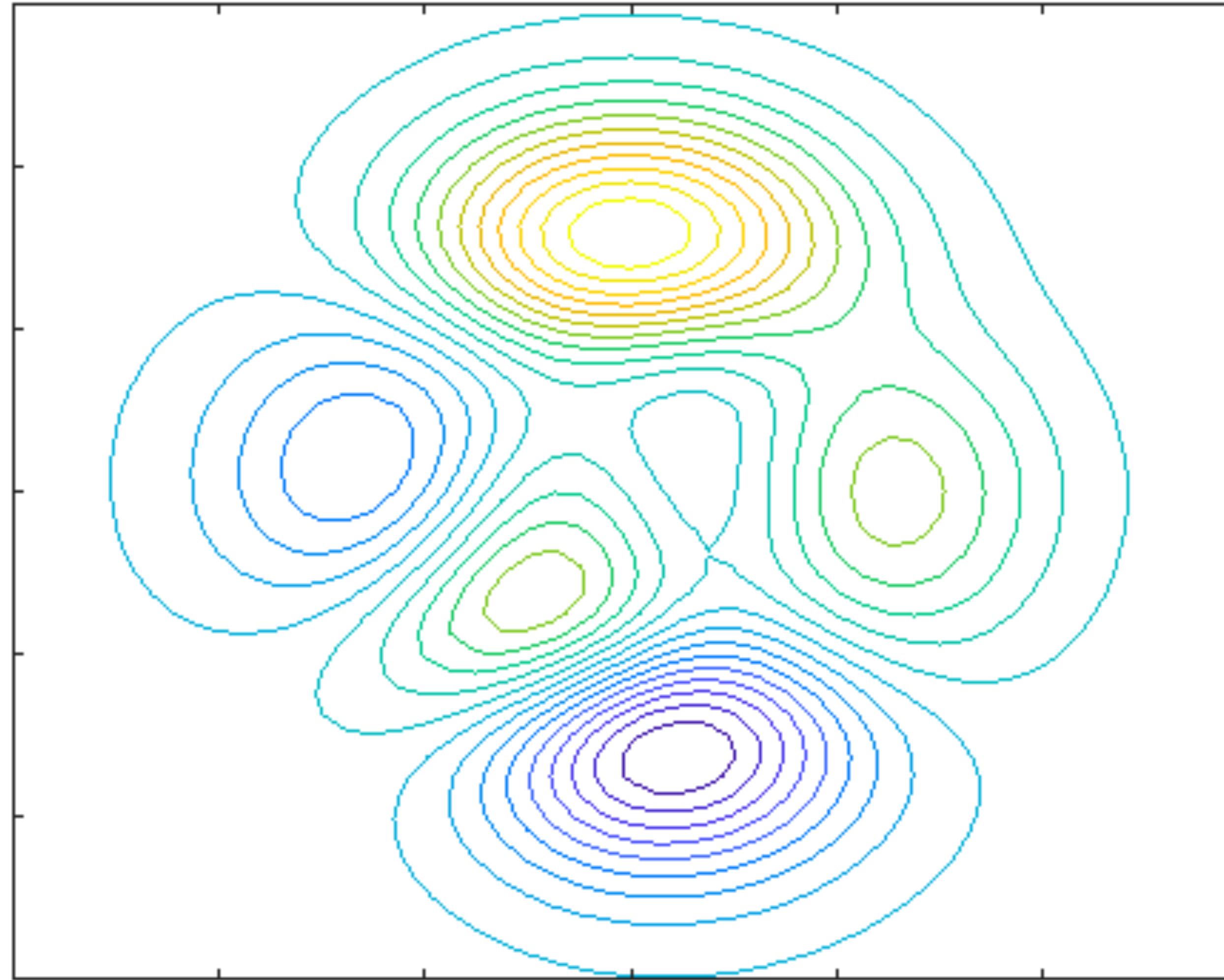
$$\begin{aligned}v_{t+1} &= \rho v_t + \nabla f(w_t) \\w_{t+1} &= w_t - \alpha v_{t+1} \\&= w_t - \alpha \rho v_t - \alpha \nabla f(w_t)\end{aligned}$$

Problems with GD (or SGD)



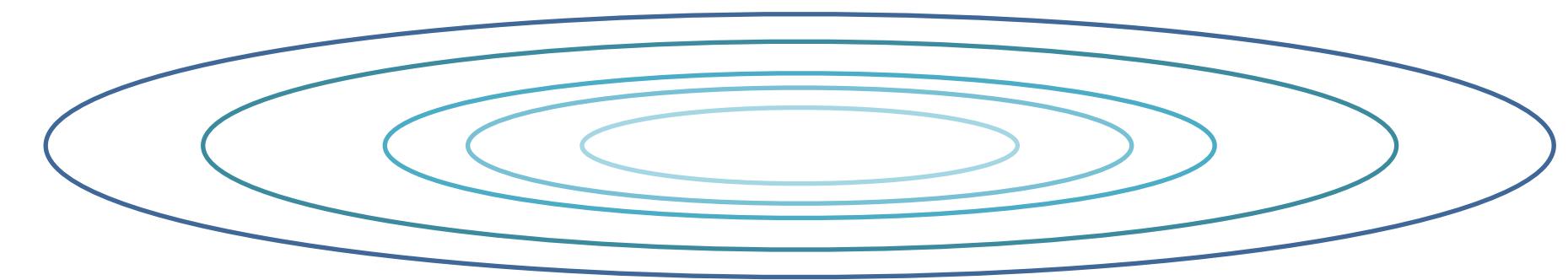
$$\begin{aligned}v_{t+1} &= \rho v_t + \nabla f(w_t) \\w_{t+1} &= w_t - \alpha v_{t+1} \\&= w_t - \alpha \rho v_t - \alpha \nabla f(w_t) \\v_{t+1} &= \rho v_t - \alpha \nabla f(w_t) \\w_{t+1} &= w_t + v_{t+1}\end{aligned}$$

Problems with GD (or SGD)



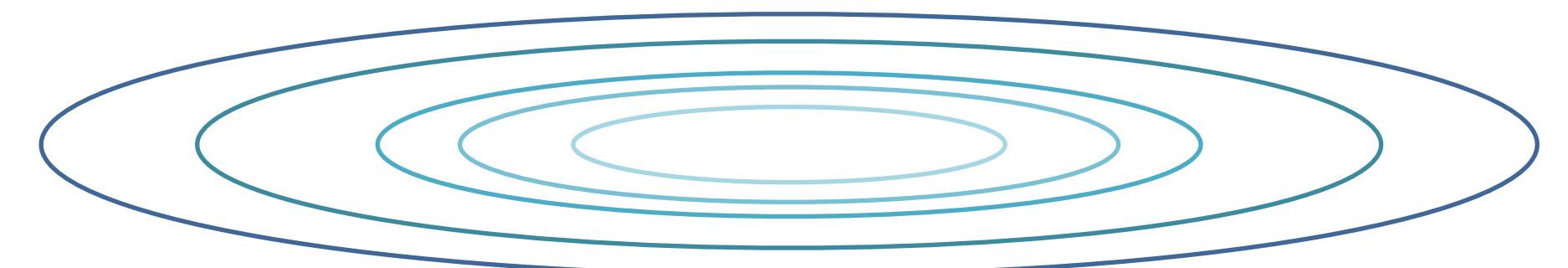
$$\begin{aligned}v_{t+1} &= \rho v_t + \nabla f(w_t) \\w_{t+1} &= w_t - \alpha v_{t+1} \\&= w_t - \alpha \rho v_t - \alpha \nabla f(w_t)\end{aligned}$$
$$\begin{aligned}v_{t+1} &= \rho v_t - \alpha \nabla f(w_t) \\w_{t+1} &= w_t + v_{t+1} \\&= w_t + \rho v_t - \alpha \nabla f(w_t)\end{aligned}$$

AdaGrad vs RmsProp



AdaGrad vs RmsProp

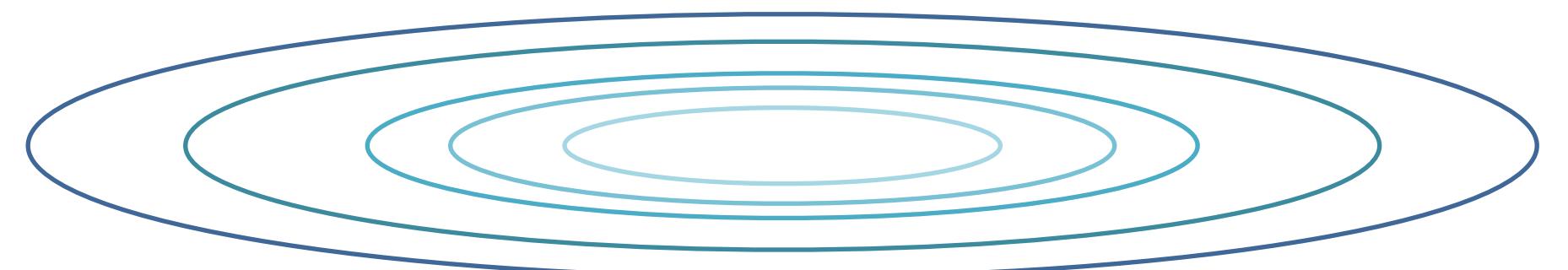
```
w = initialize_weights()
for t in range(num_steps):
    dw = compute_gradient(loss_fn, data, w)
    w -= learning_rate * dw
```



AdaGrad vs RmsProp

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

AdaGrad

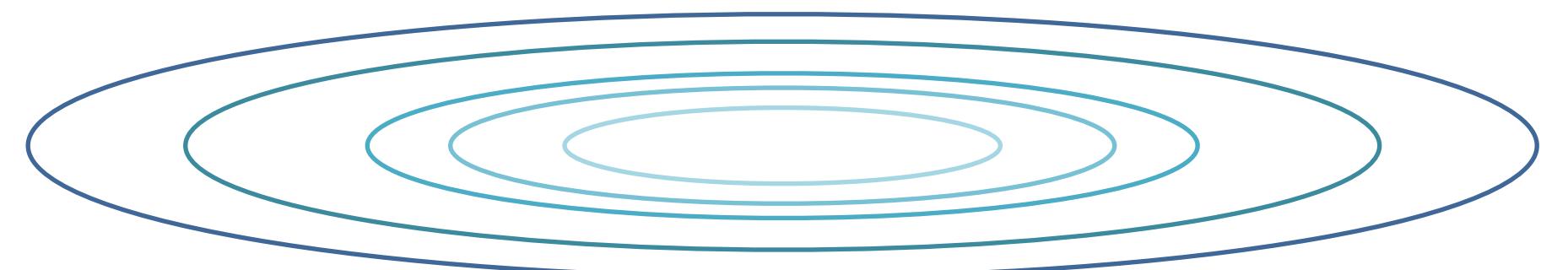


AdaGrad vs RmsProp

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

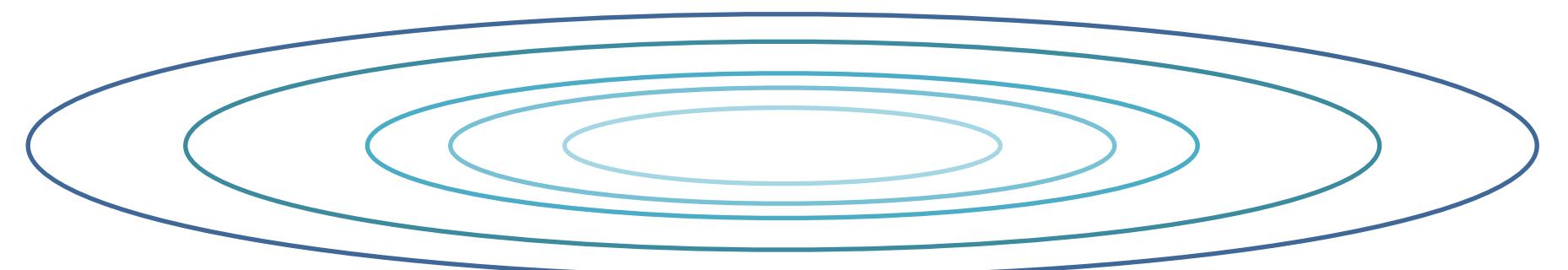
AdaGrad

$$\text{gradSquared} += \mathbf{dw} * \mathbf{dw}$$



AdaGrad vs RmsProp

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



AdaGrad

$$\text{gradSquared} += \mathbf{dw} * \mathbf{dw}$$

$$\mathbf{w} = \mathbf{w} - \alpha \mathbf{dw} / (\sqrt{\text{gradSquared}} + \epsilon)$$

AdaGrad vs RmsProp

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

AdaGrad

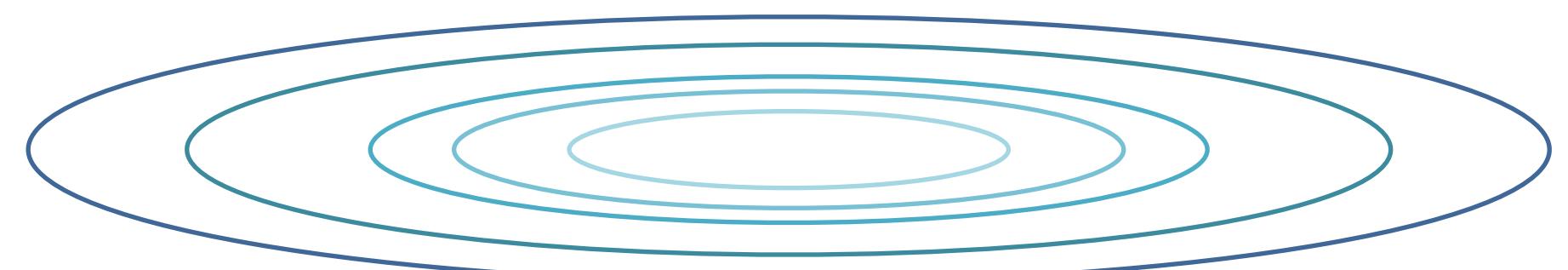
$$\text{gradSquared} += \mathbf{dw} * \mathbf{dw}$$

$$\mathbf{w} = \mathbf{w} - \alpha \mathbf{dw} / (\sqrt{\text{gradSquared}} + \epsilon)$$

RmsProp

$$\text{gradSquared} = \gamma \text{gradSquared} + (1 - \gamma) \mathbf{dw} * \mathbf{dw}$$

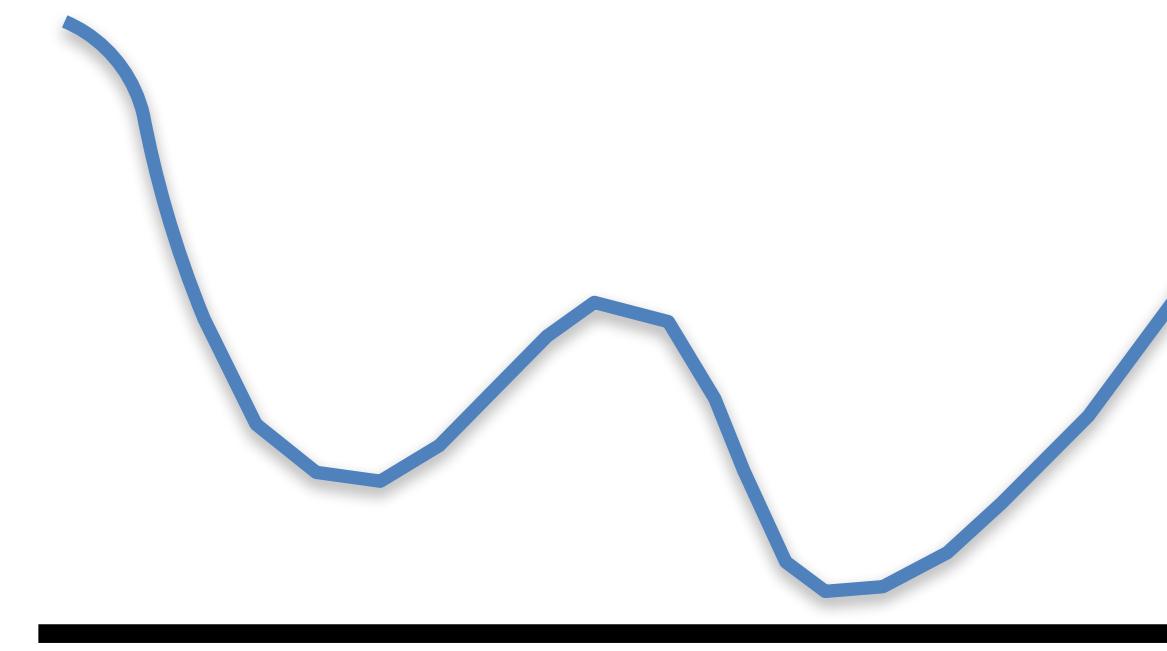
$$\mathbf{w} = \mathbf{w} - \alpha \mathbf{dw} / (\sqrt{\text{gradSquared}} + \epsilon)$$



Adam (most common choice)

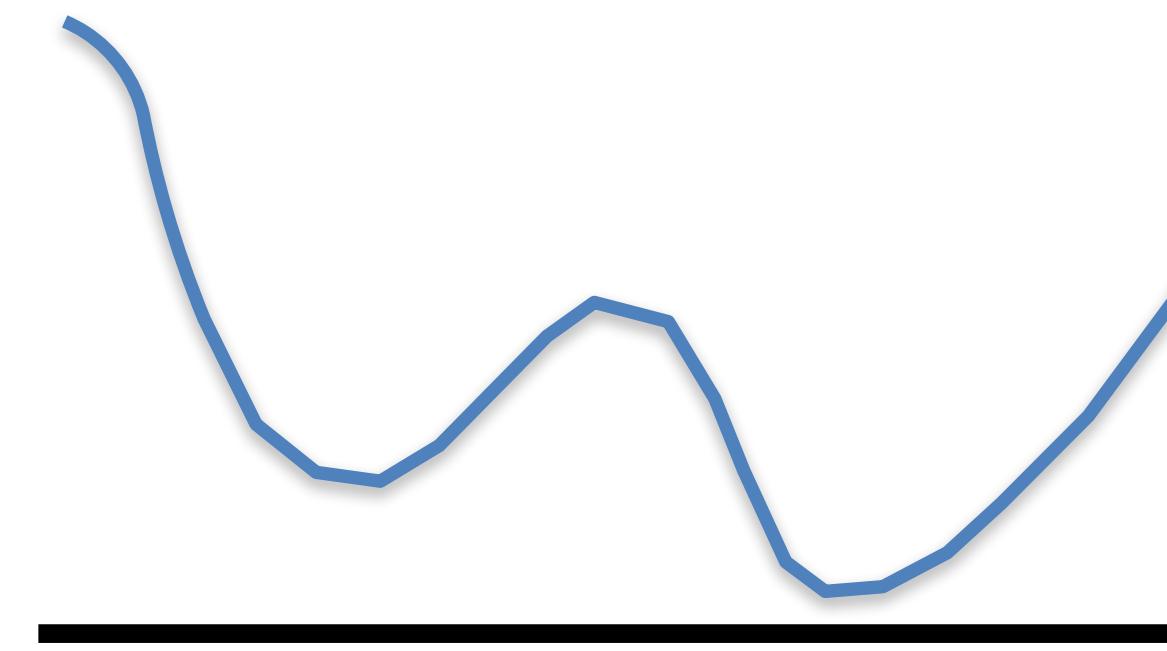
- A bit more than AdaGrad + RmsProp

First versus Second Order Methods



$$F(w) \approx F(w_0) + (w - w_0)^T \nabla_w F(w_0) + \frac{1}{2} (w - w_0)^T H_w (w - w_0)$$

First versus Second Order Methods



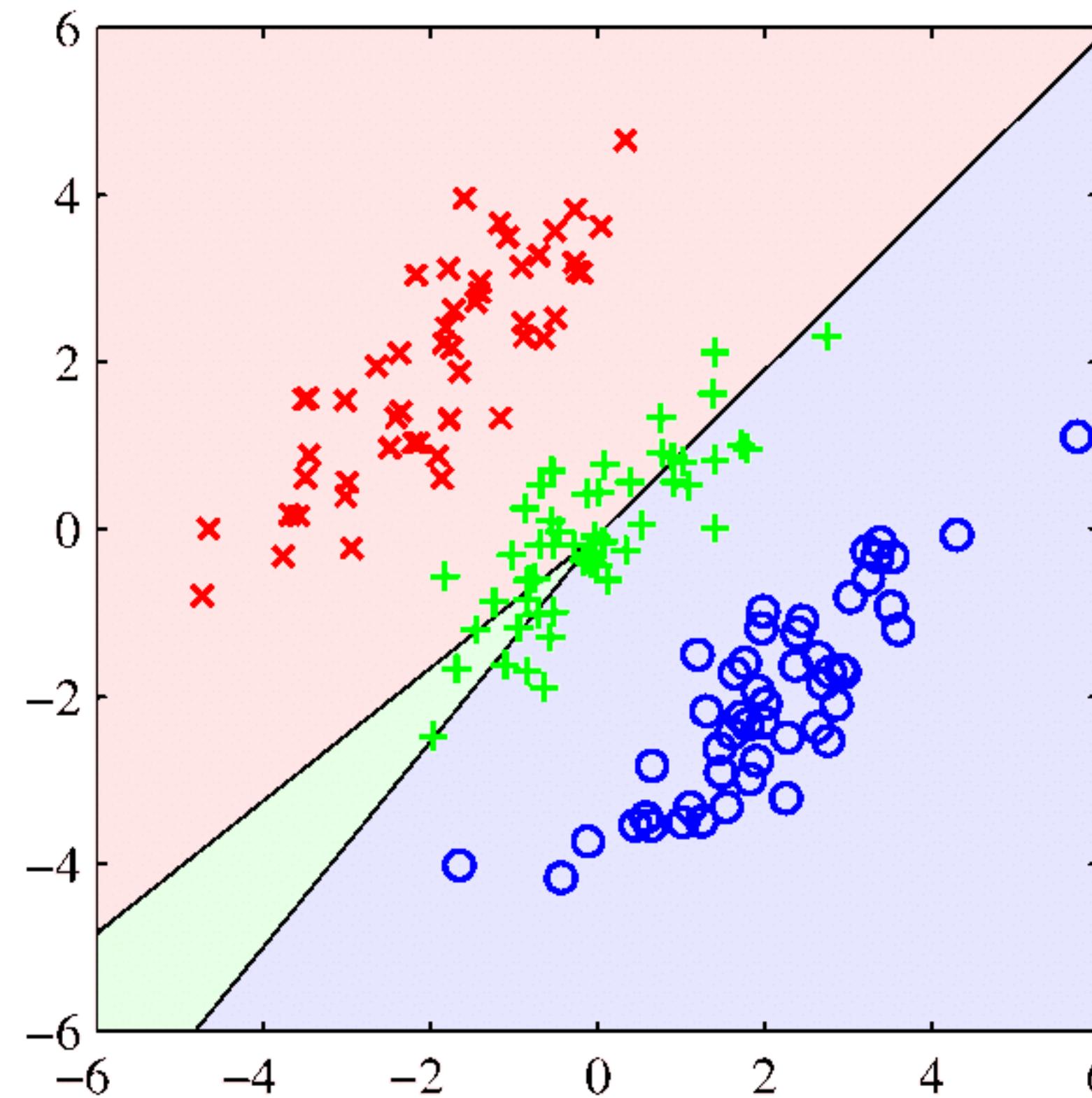
$$F(w) \approx F(w_0) + (w - w_0)^T \nabla_w F(w_0) + \frac{1}{2} (w - w_0)^T H_w (w - w_0)$$

$$w^* = w_0 - H_w^{-1}(w_0) \nabla_w F(w_0)$$

Logistic vs Linear Regression, $n>2$ classes

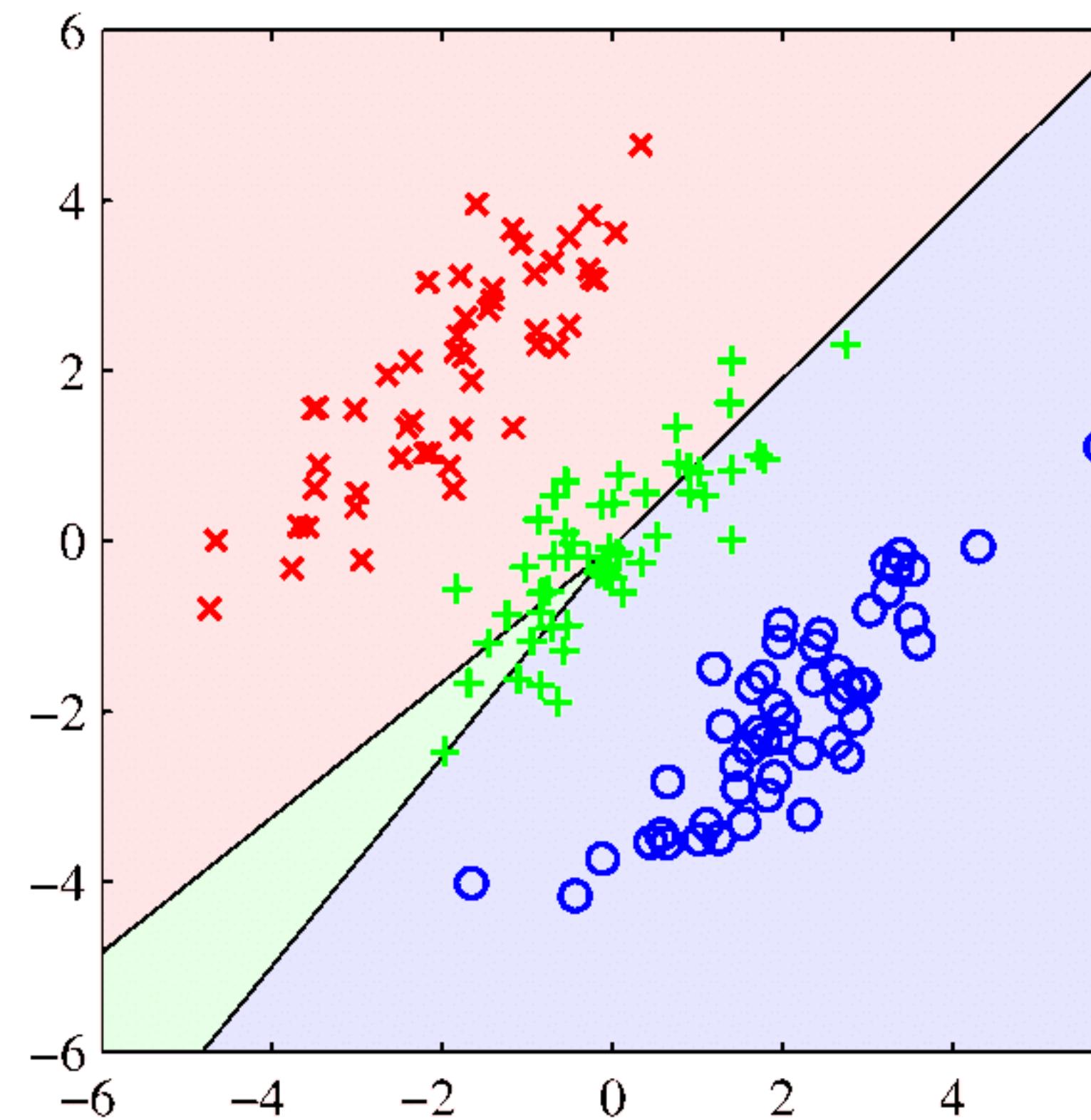
Logistic vs Linear Regression, $n > 2$ classes

Linear regression

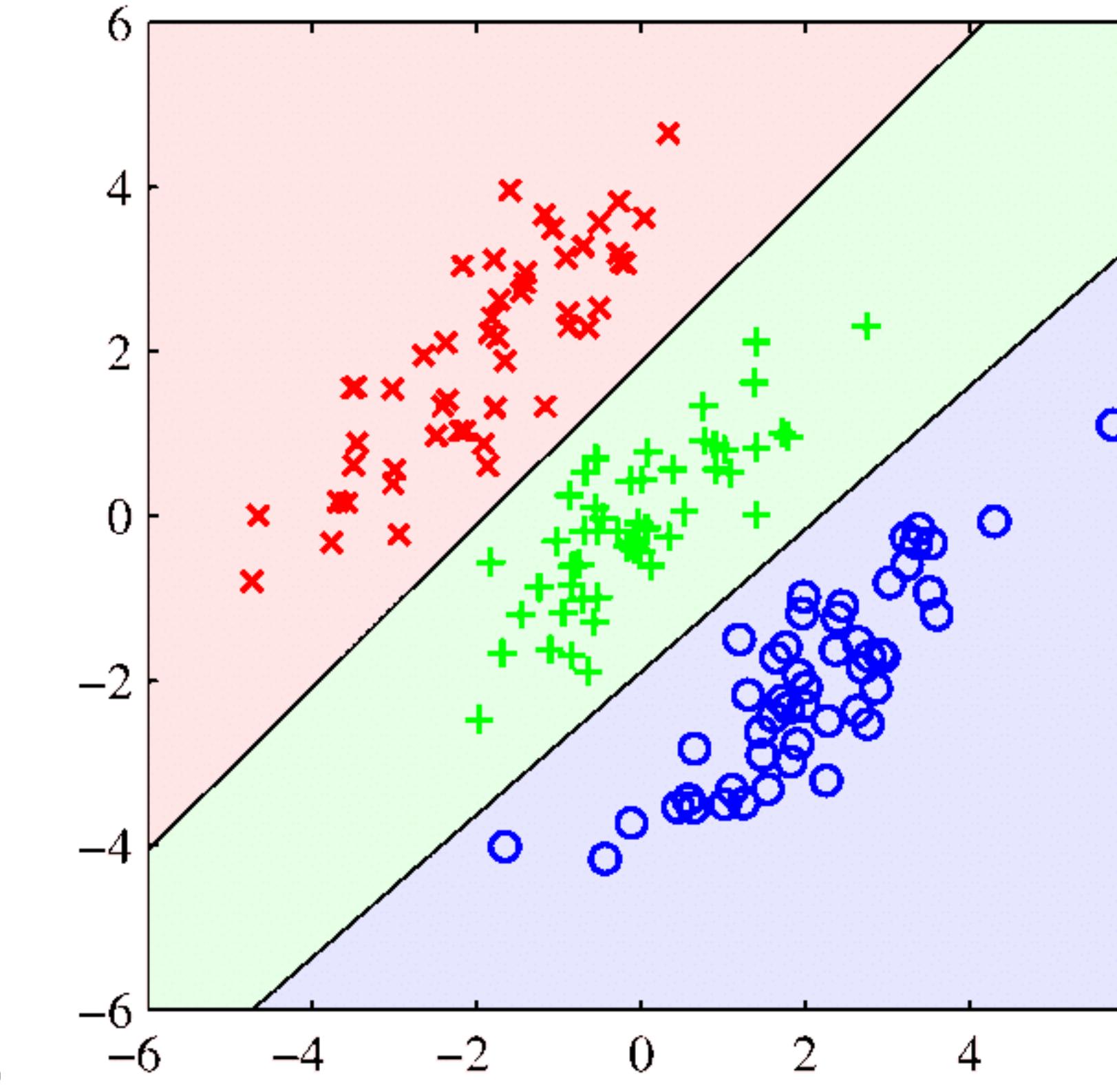


Logistic vs Linear Regression, $n > 2$ classes

Linear regression

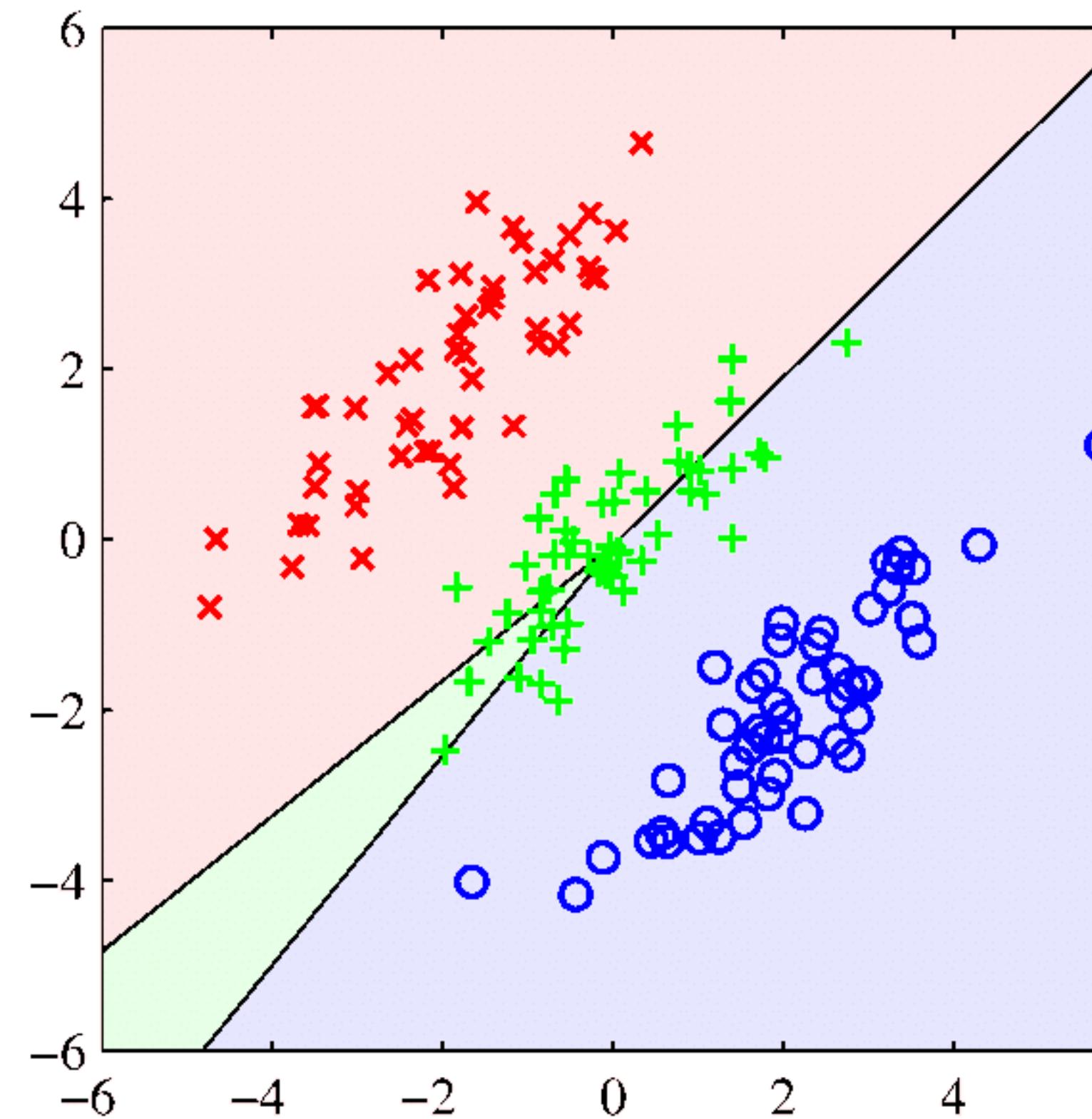


Logistic regression

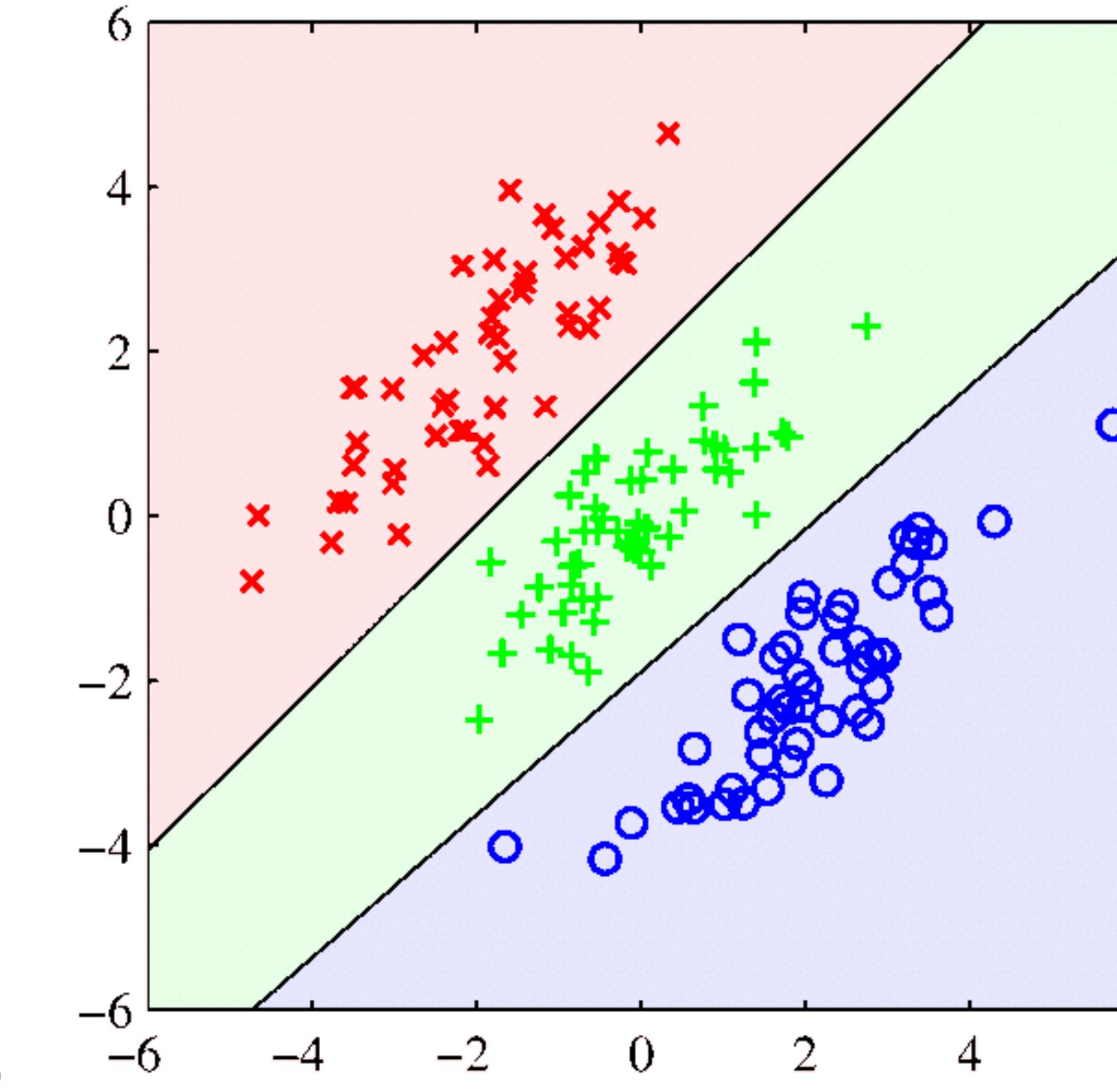


Logistic vs Linear Regression, $n > 2$ classes

Linear regression



Logistic regression



Logistic regression does not exhibit the masking problem

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

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

Particular choice of form of f:

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

Form of posterior distribution

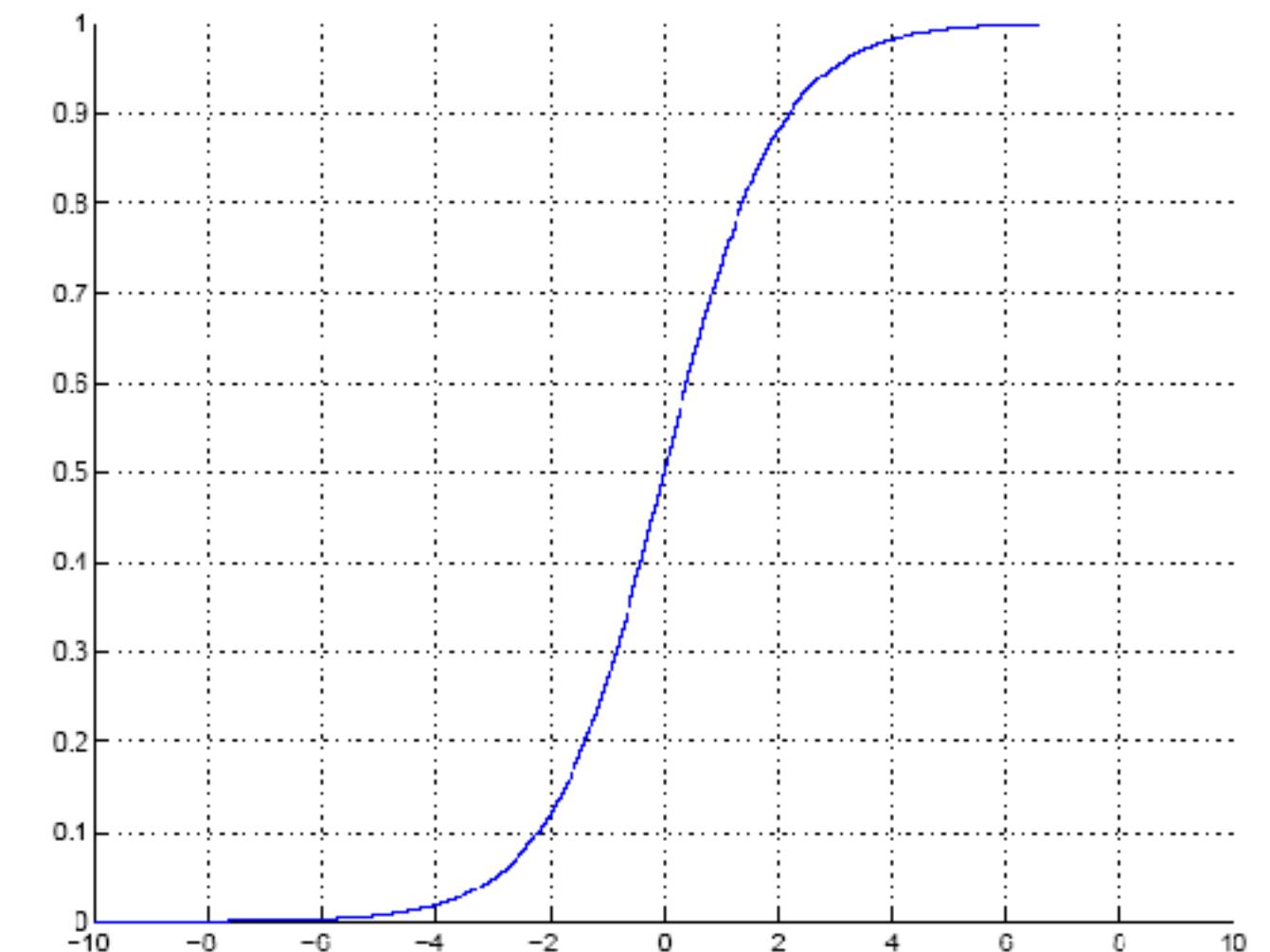
Bernoulli-type conditional distribution

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

Particular choice of form of f:

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

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



Form of posterior distribution

Bernoulli-type conditional distribution

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

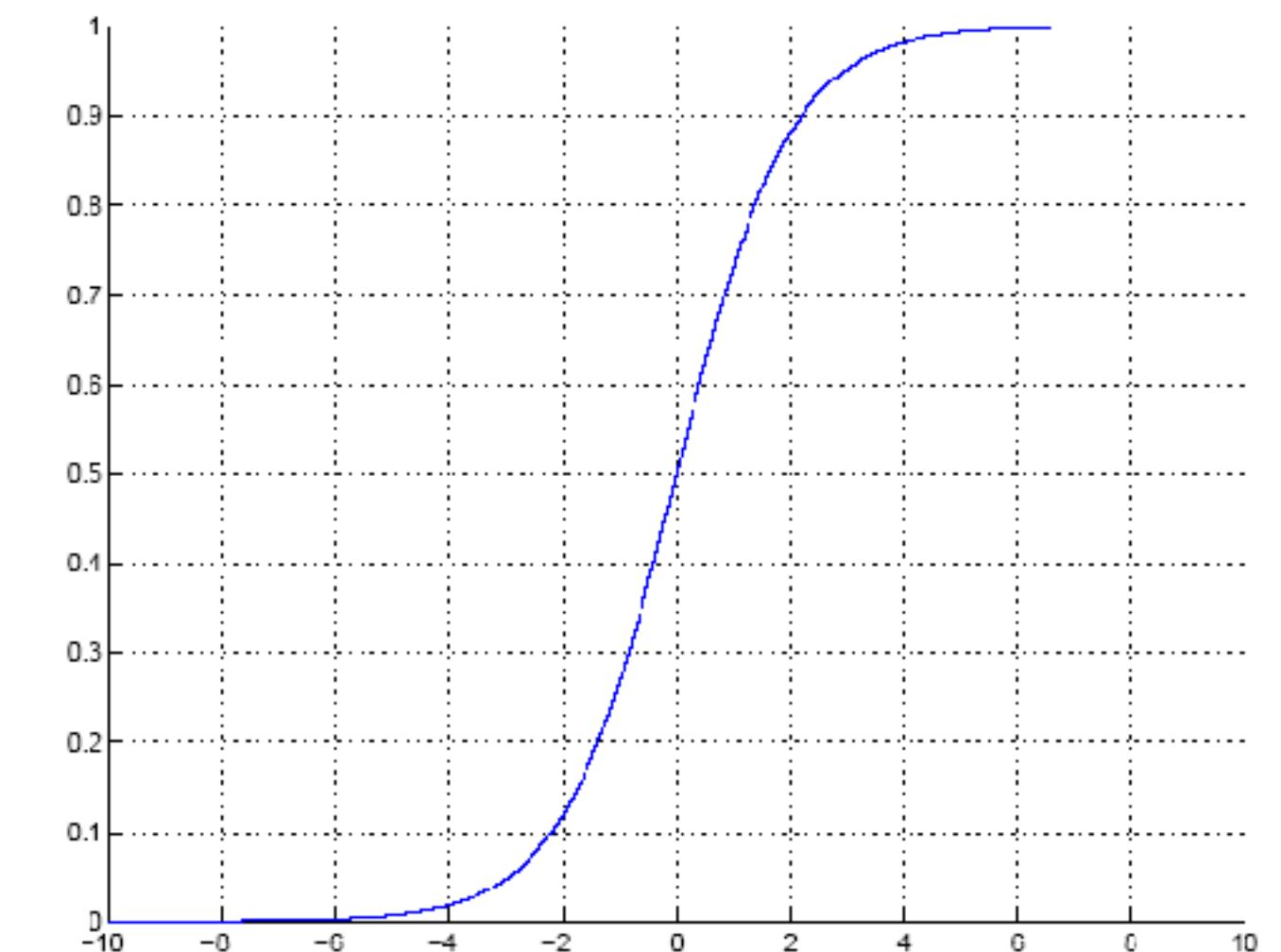
Particular choice of form of f :

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

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

“squashing function”:

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



Form of posterior distribution

Bernoulli-type conditional distribution

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

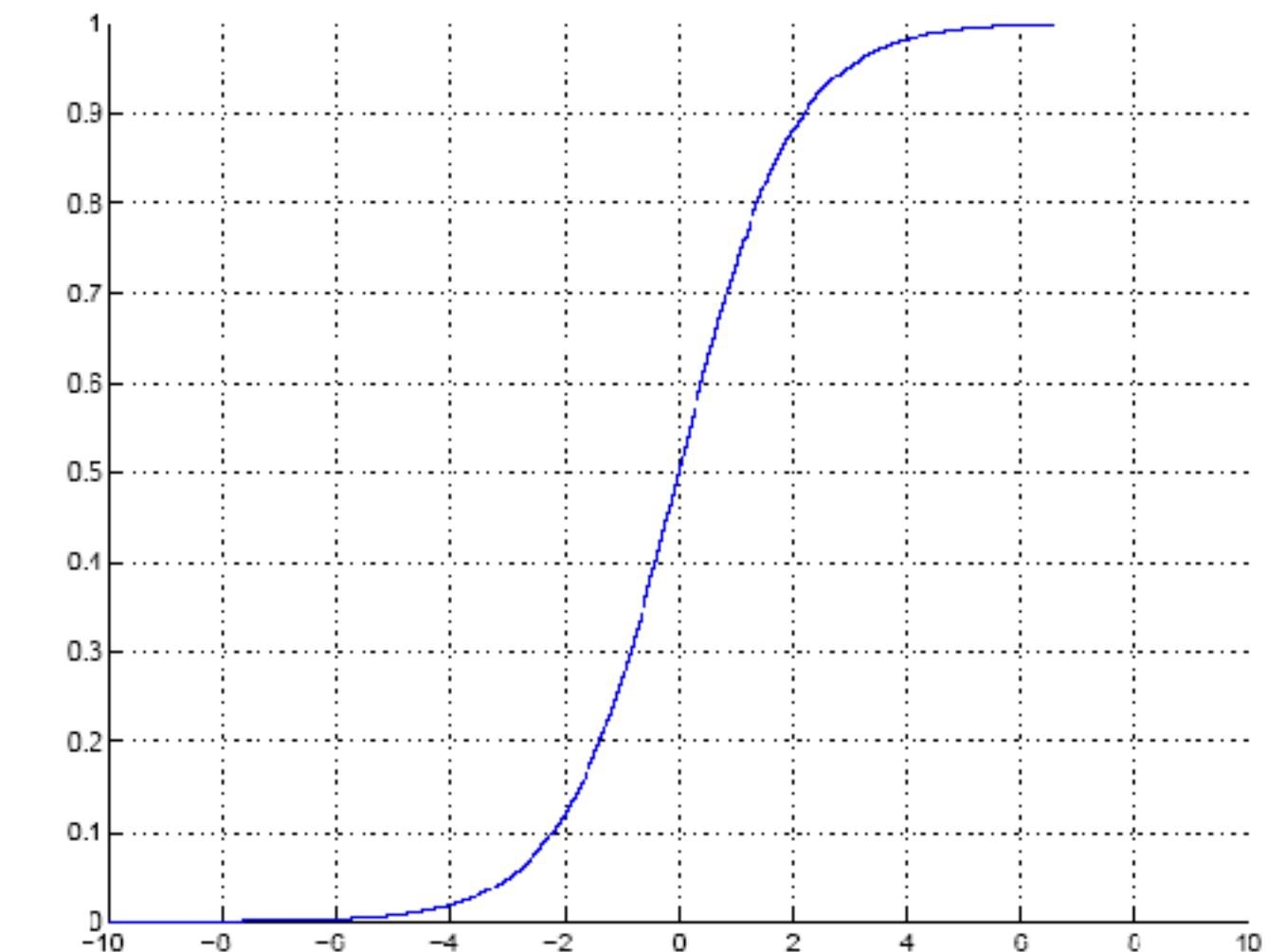
Particular choice of form of f :

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

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

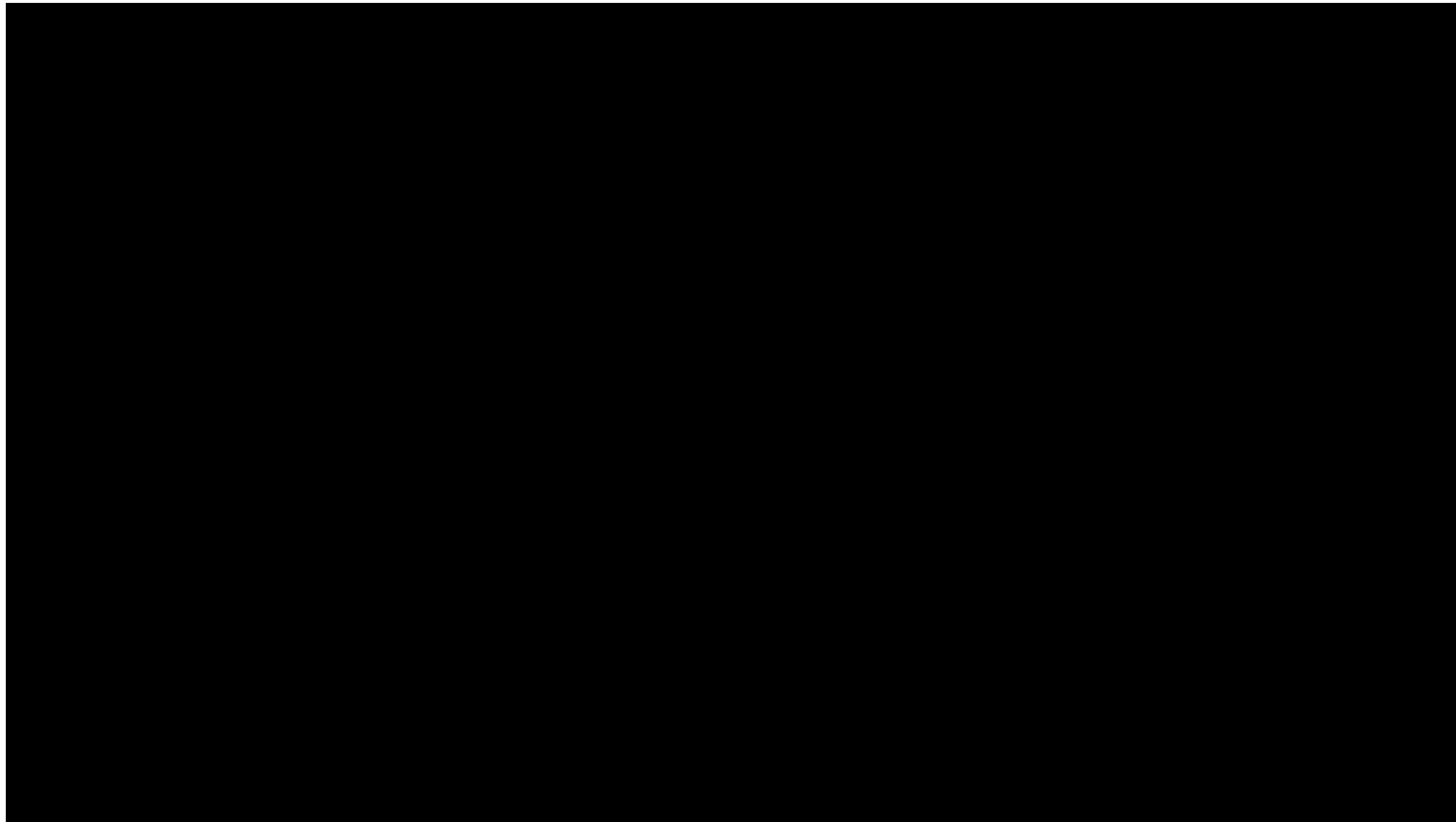
“squashing function”:

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



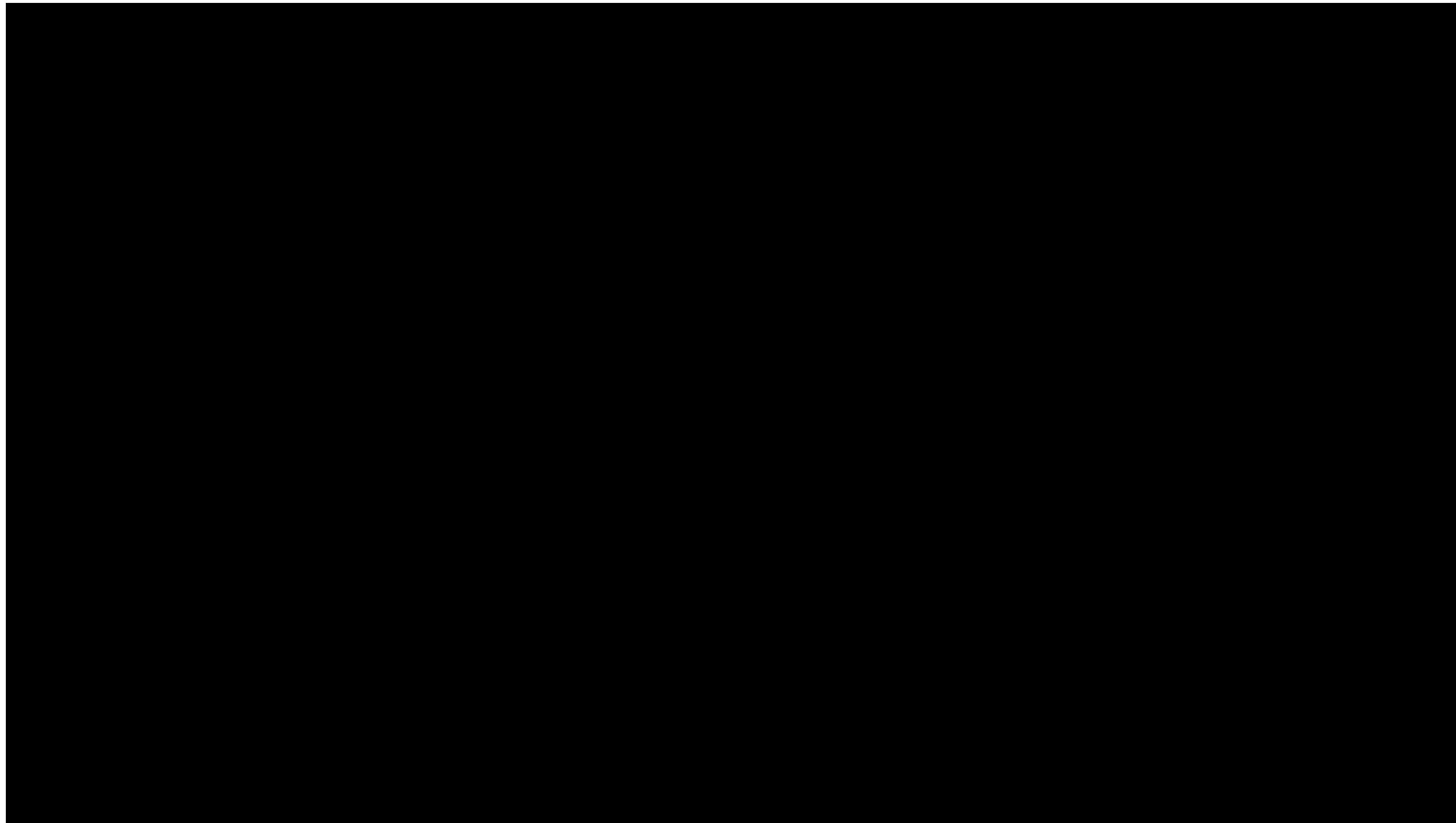
Negative LogLikelihood

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

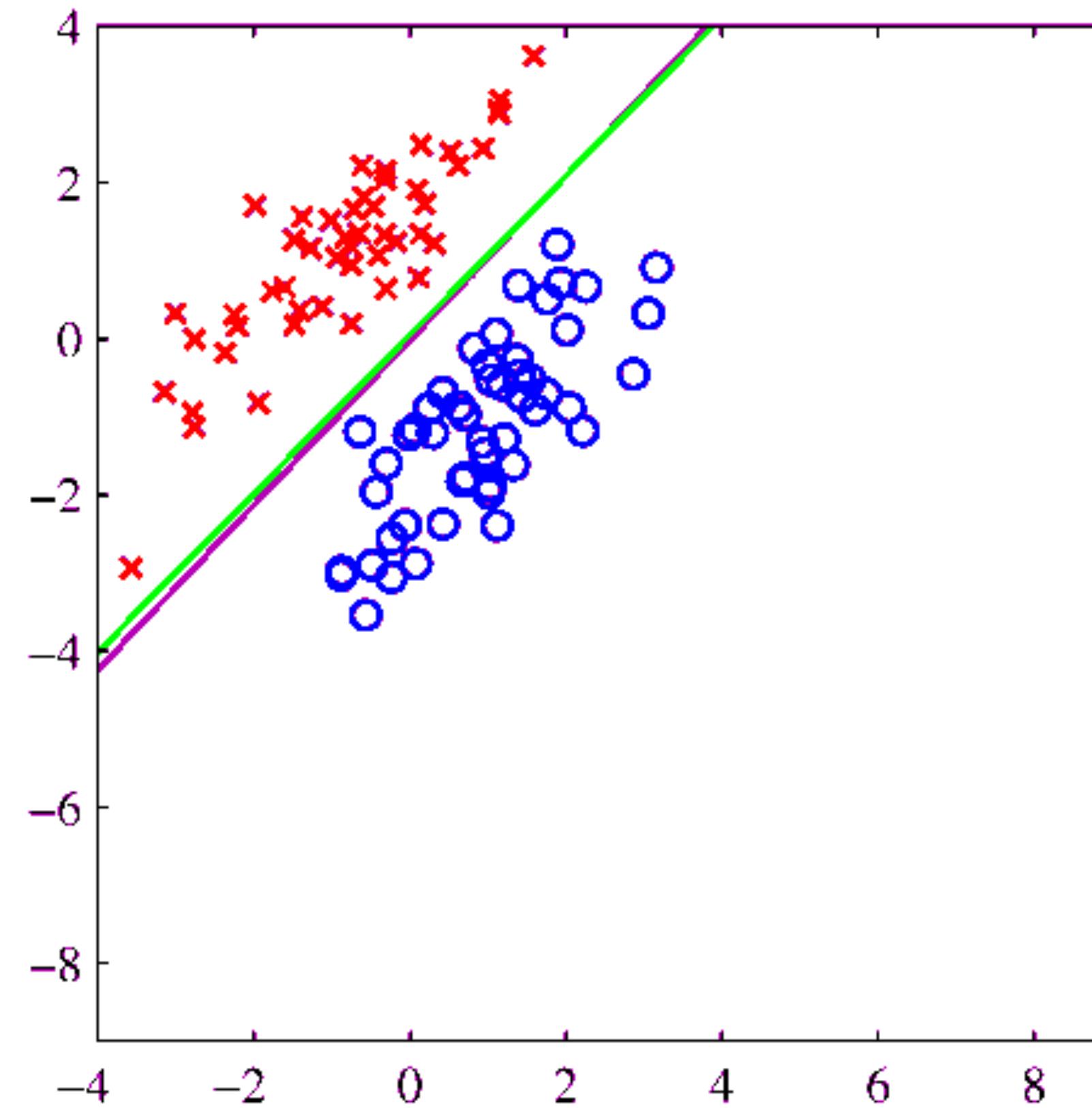


Negative LogLikelihood

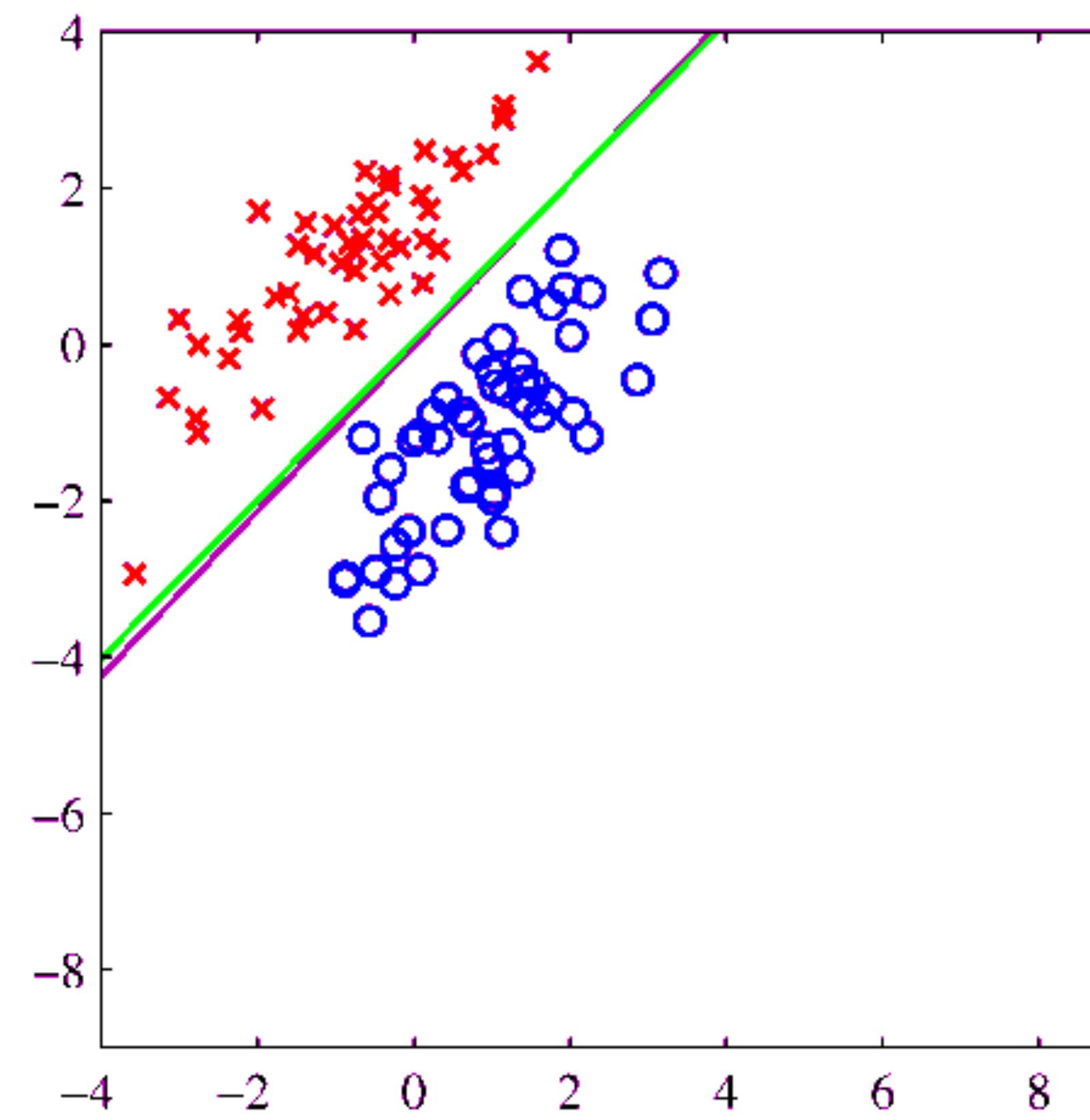
$$\log \mathbb{P}(\mathbf{D}|\theta) = \sum_i (y_i \log \hat{y}_{\theta,i} + (1 - y_i) \log(1 - \hat{y}_{\theta,i}))$$



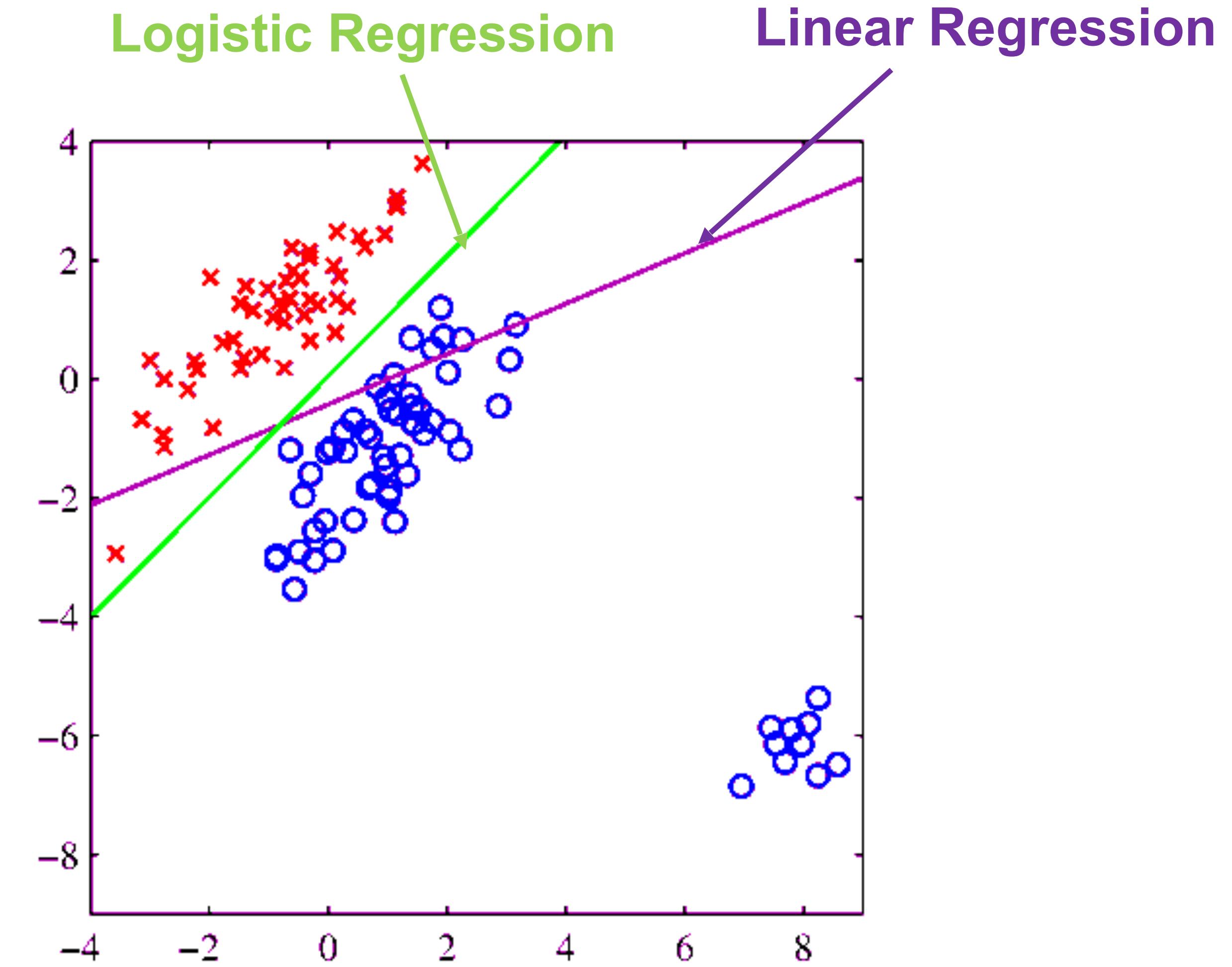
Logistic vs Linear Regression



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

Multiple Classes & Linear Regression

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

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

Matrix notation:

Multiple Classes & Linear Regression

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

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

Matrix notation:

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

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:

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

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

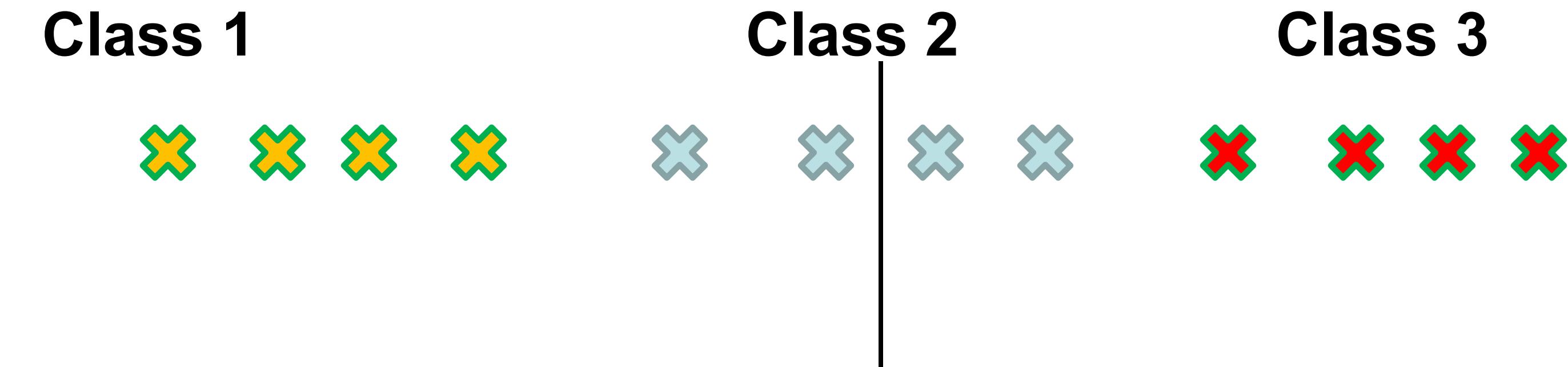
Loss function:

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

Least squares fit (decouples per class):

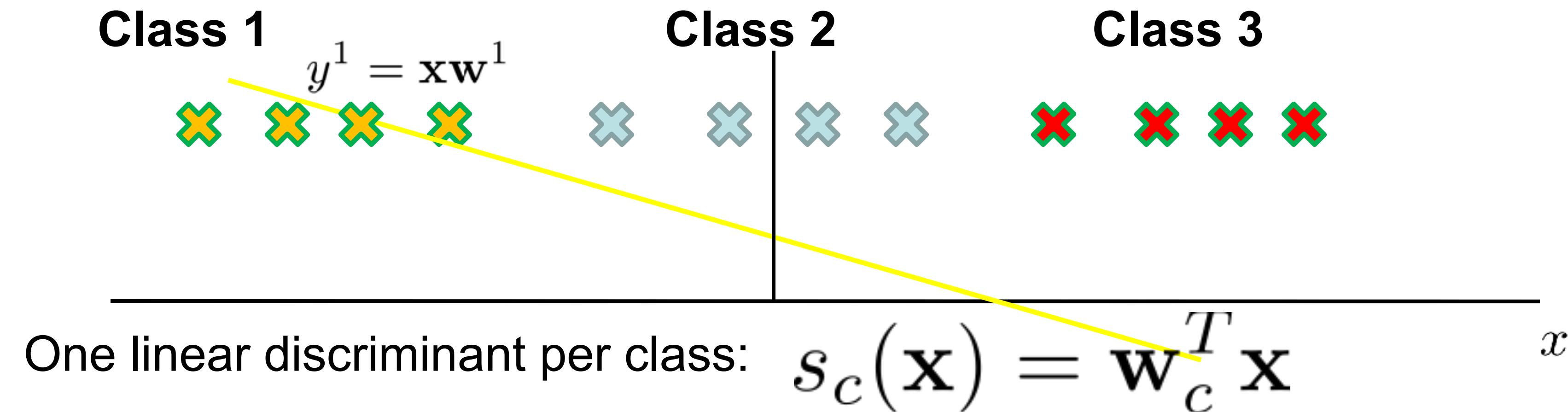
$$\mathbf{w}_c^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}_c$$

Linear Regression Masking Problem

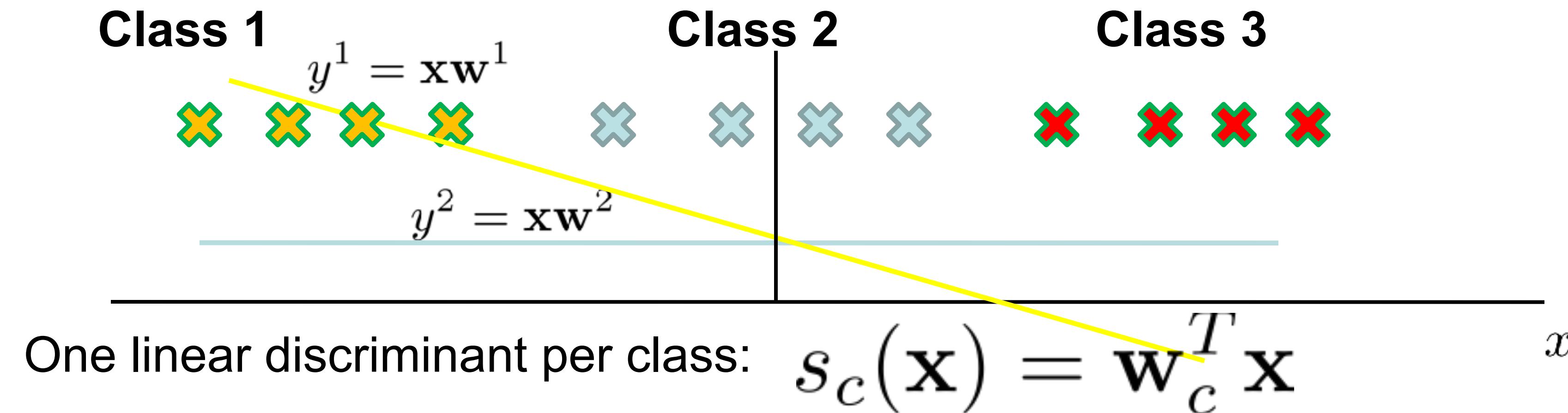


One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

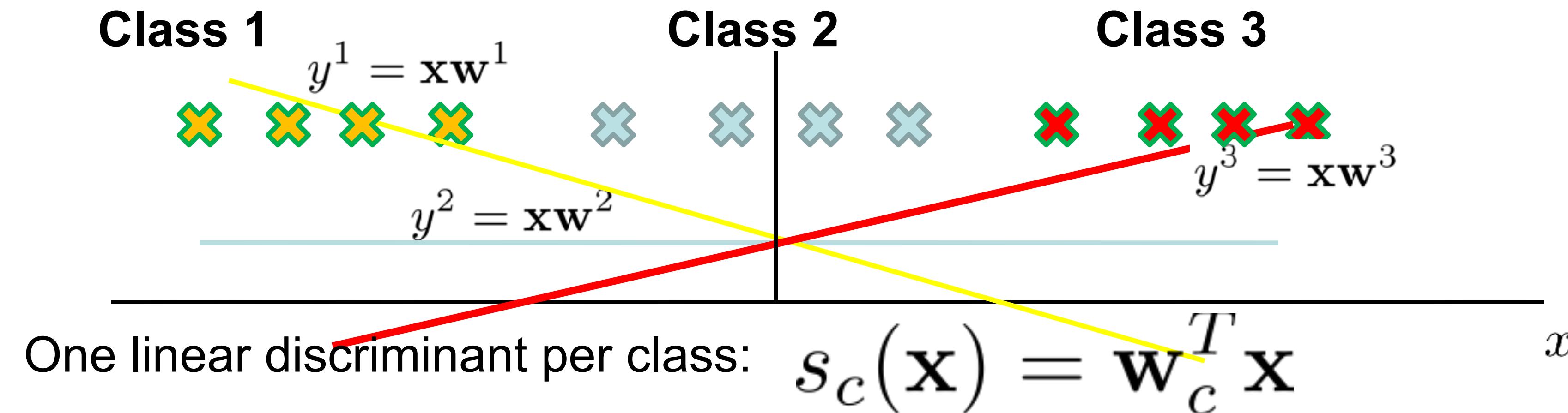
Linear Regression Masking Problem



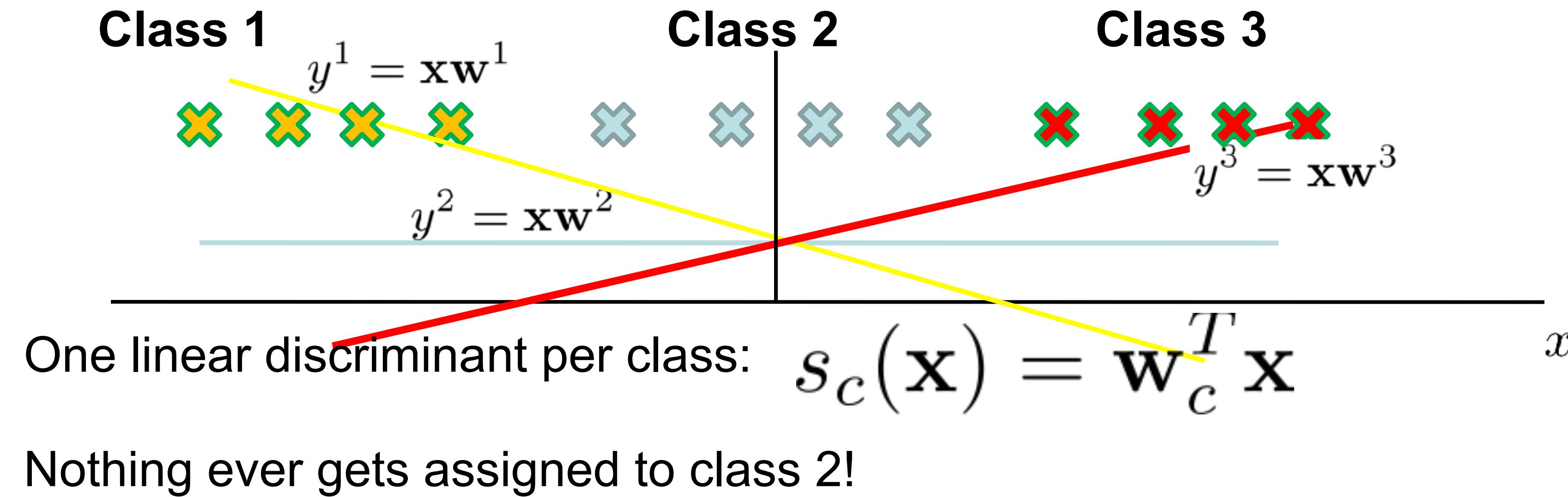
Linear Regression Masking Problem



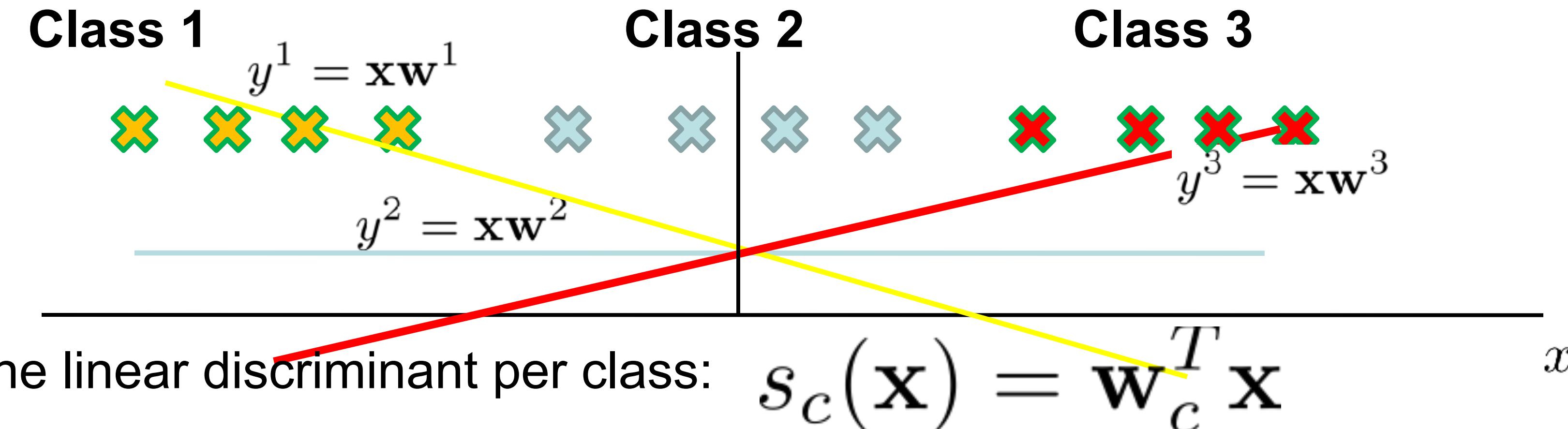
Linear Regression Masking Problem



Linear Regression Masking Problem

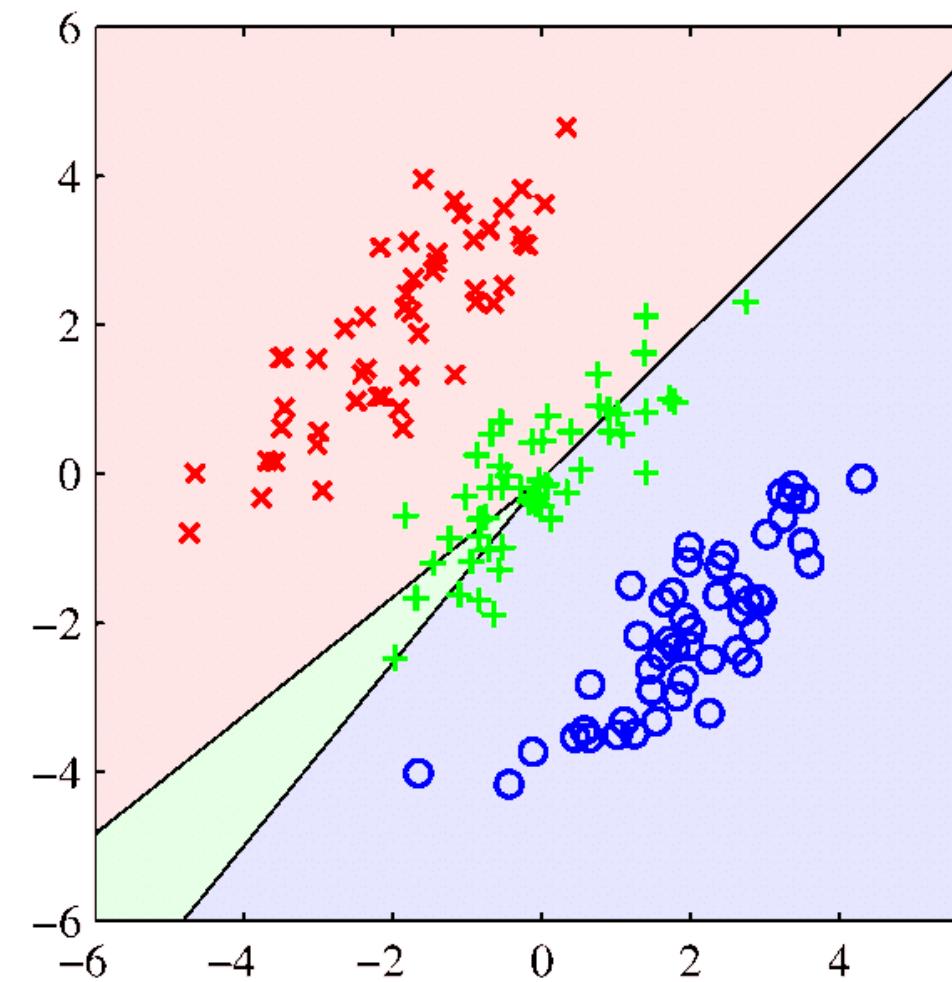


Linear Regression Masking Problem



Nothing ever gets assigned to class 2!

2D version:



Multiple classes & Logistic regression

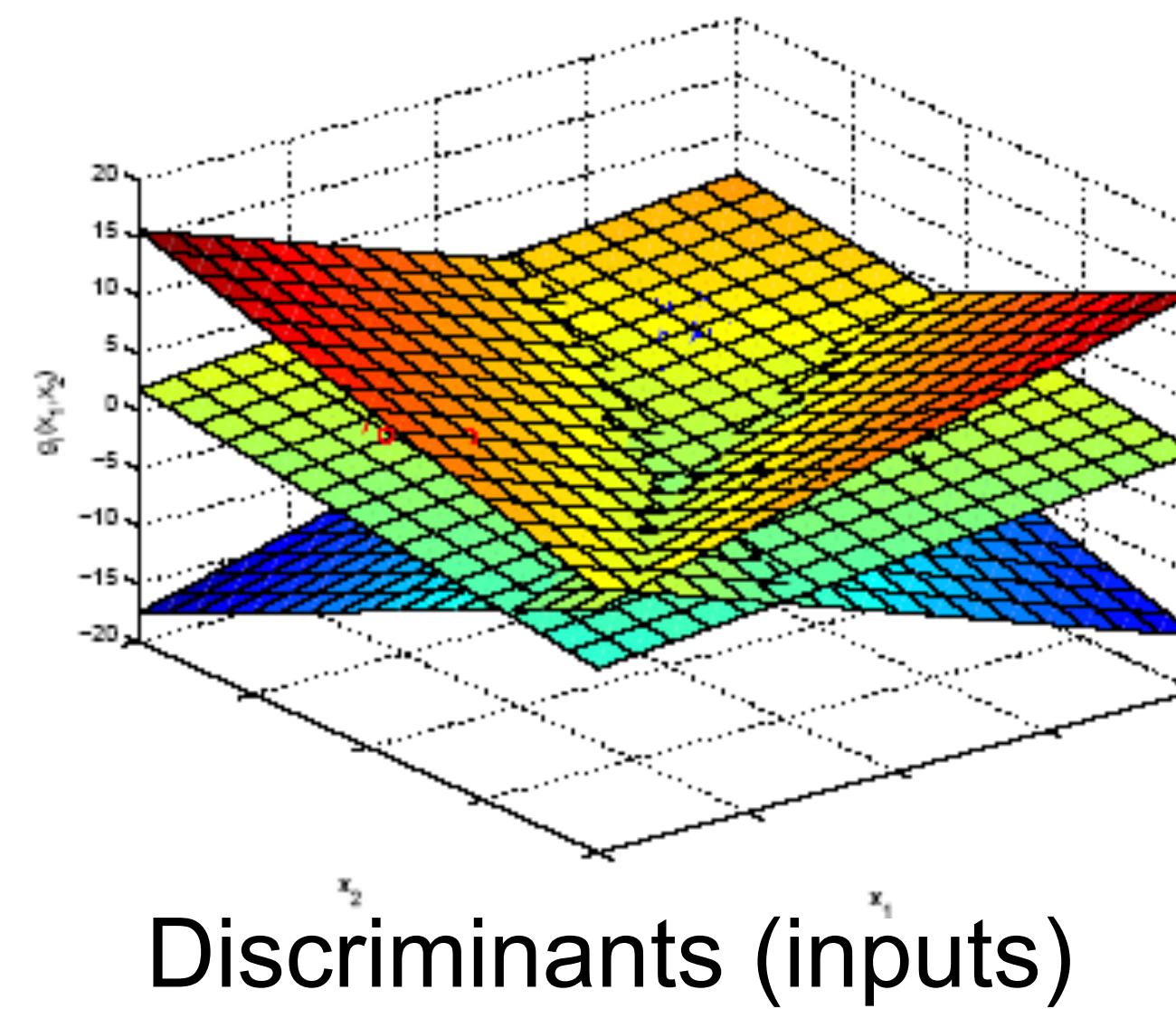
Soft maximum (softmax) of competing classes:

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

Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

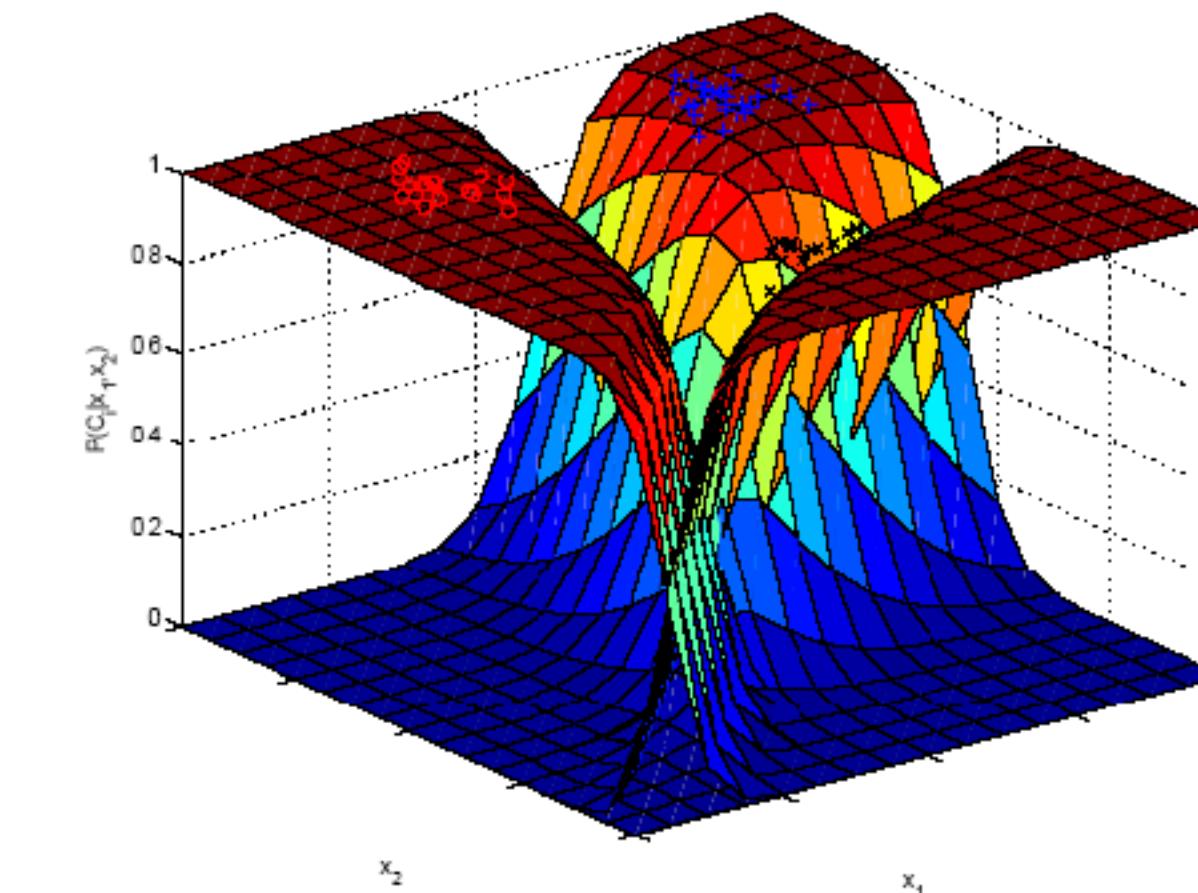
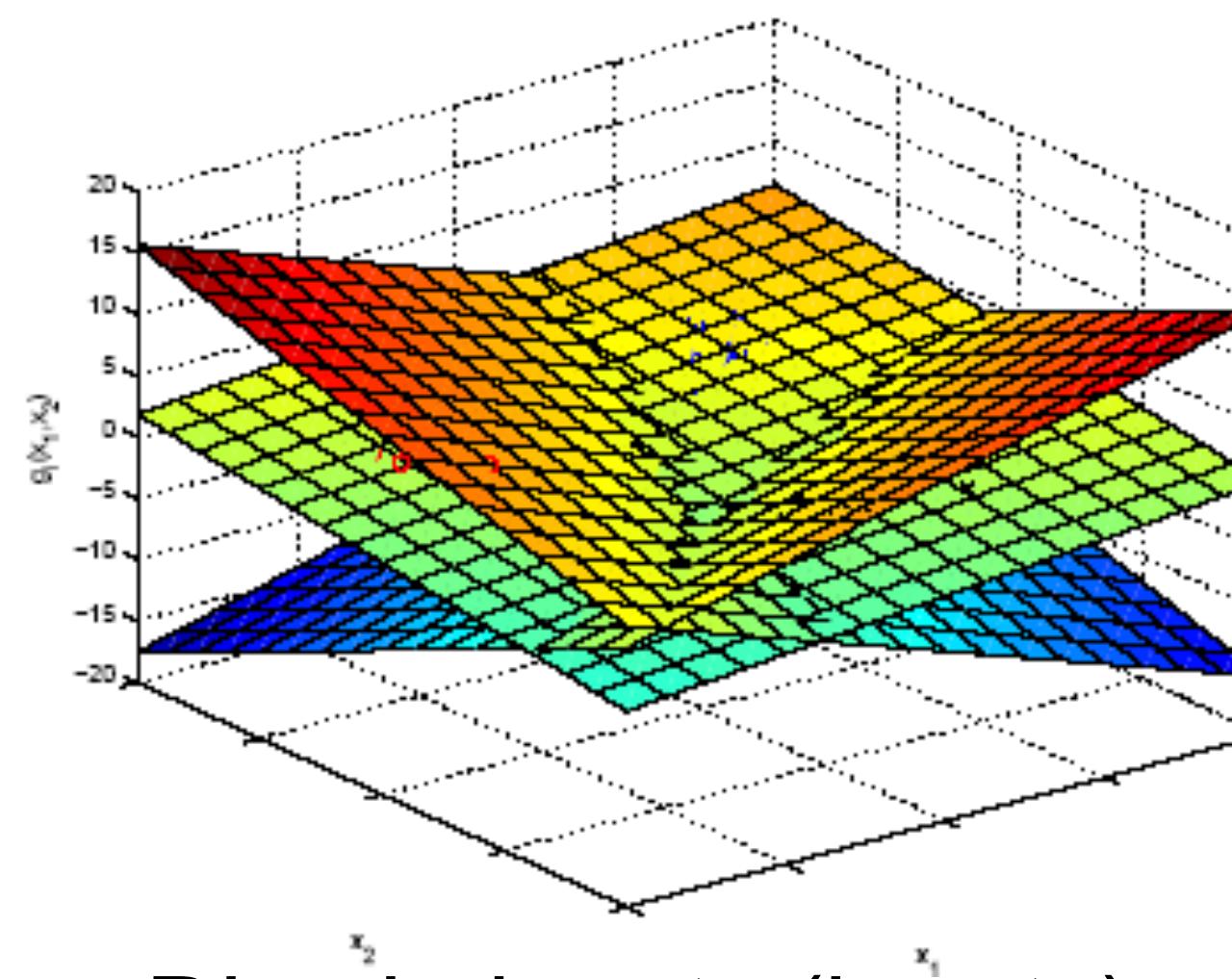
$$P(y = c|\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^C \exp(\mathbf{w}_{c'}^T \mathbf{x})} \doteq g_c(\mathbf{x}, \mathbf{W})$$



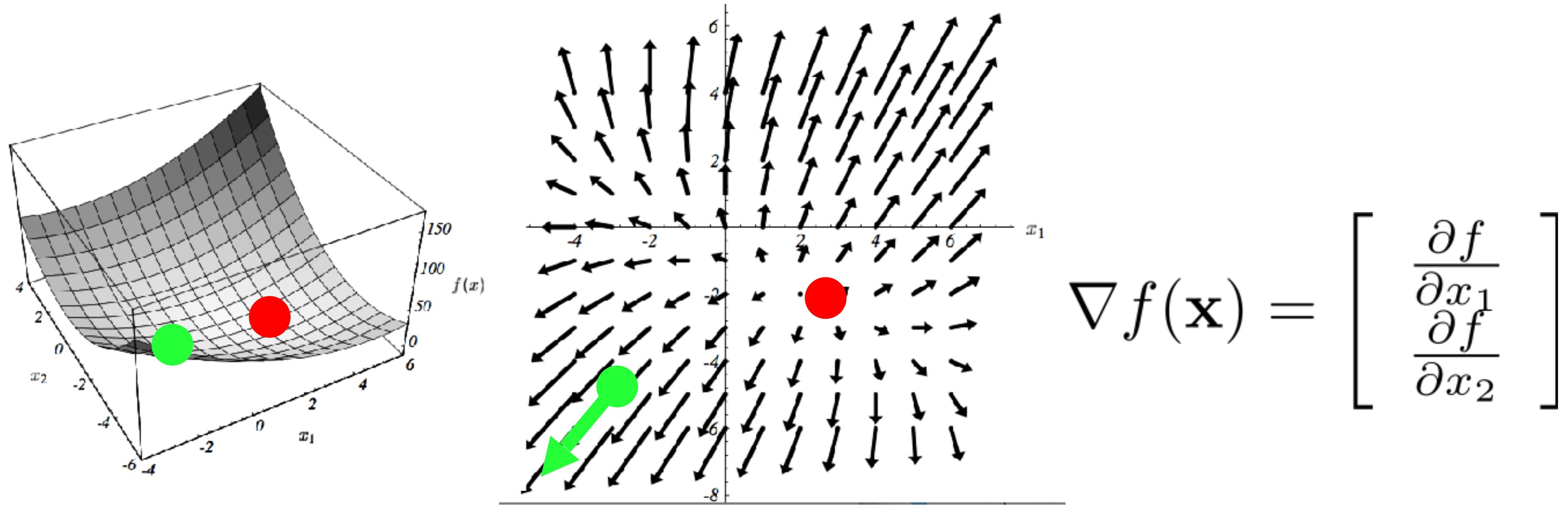
Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

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

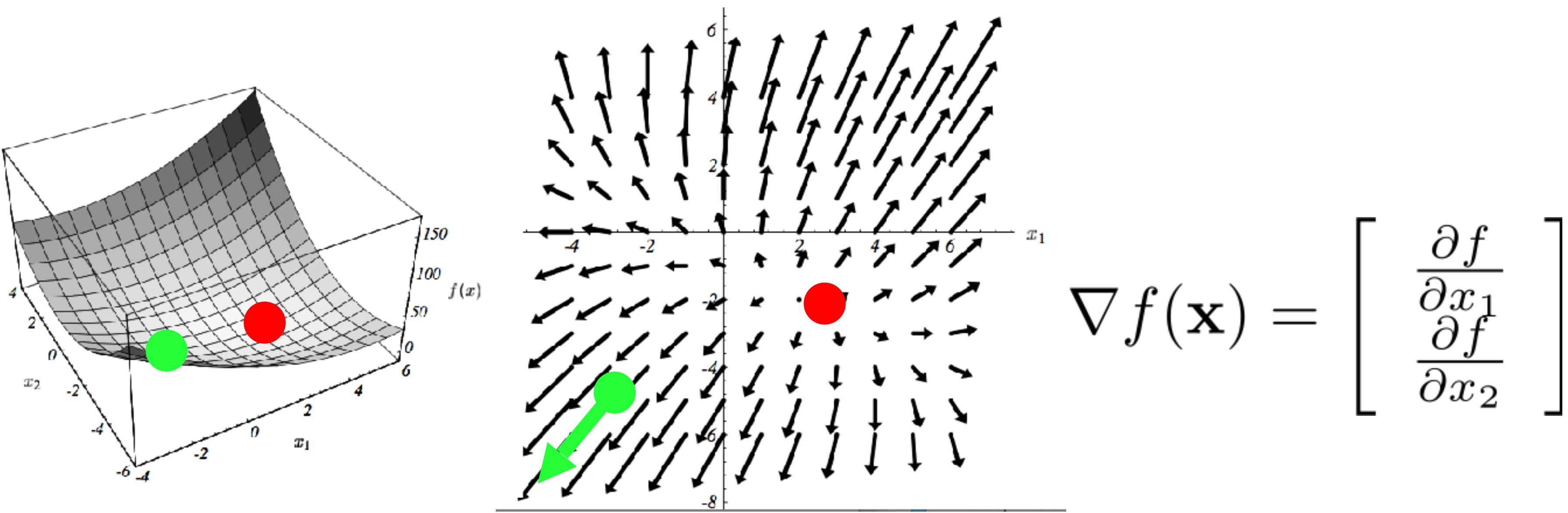


Gradient Descent Minimization



Fact: gradient at any point gives direction of fastest increase

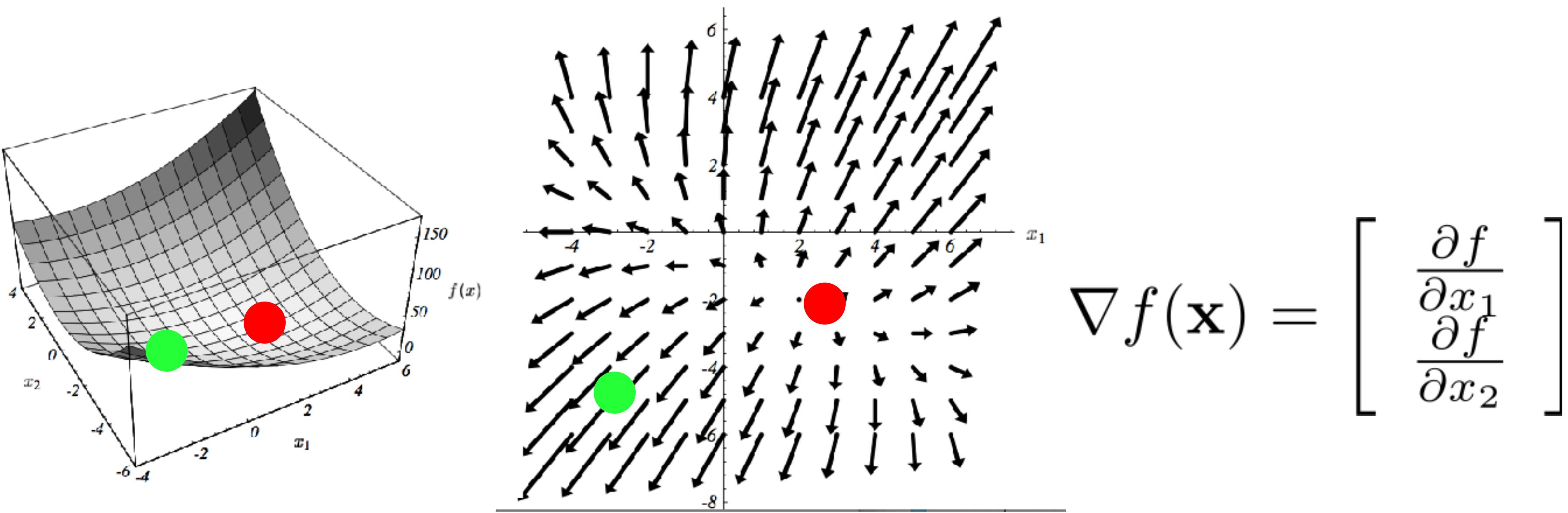
Gradient Descent Minimization



Fact: gradient at any point gives direction of fastest increase

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

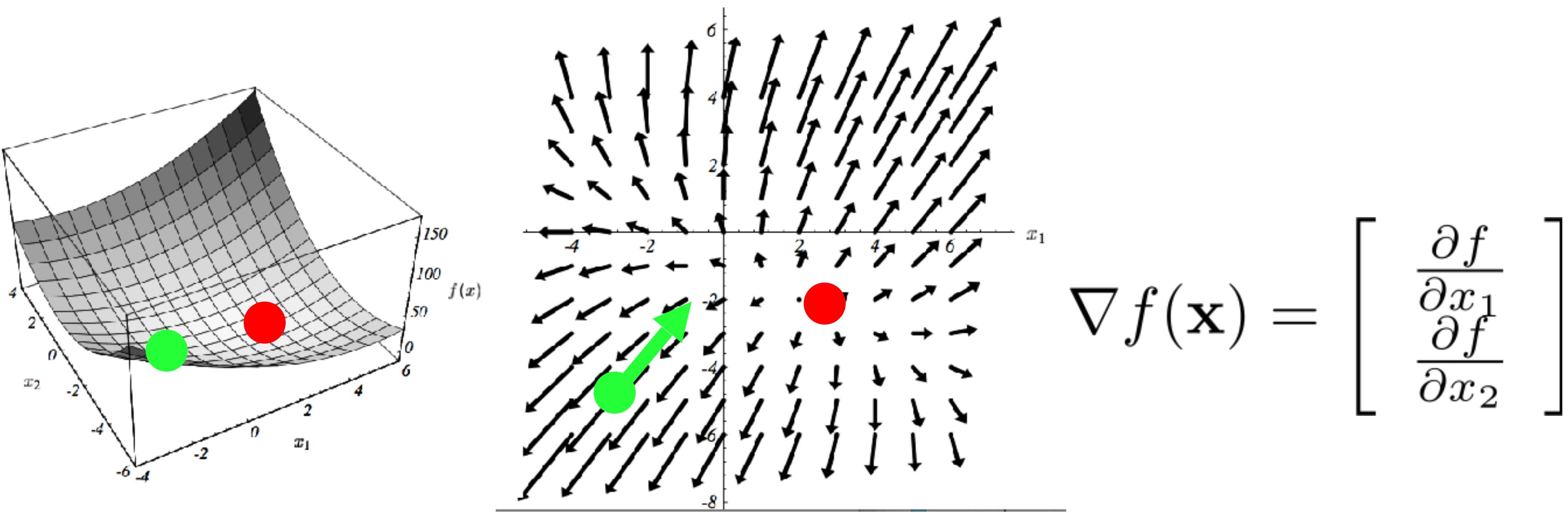
Gradient Descent Minimization



Fact: gradient at any point gives direction of fastest increase

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

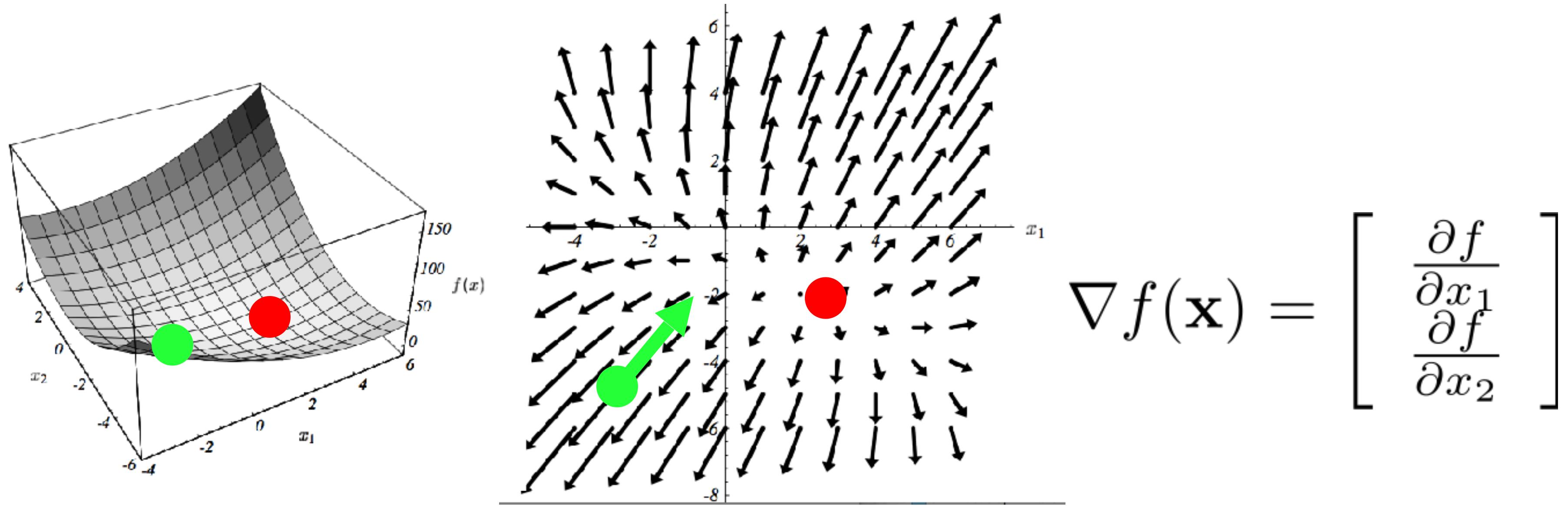
Gradient Descent Minimization



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

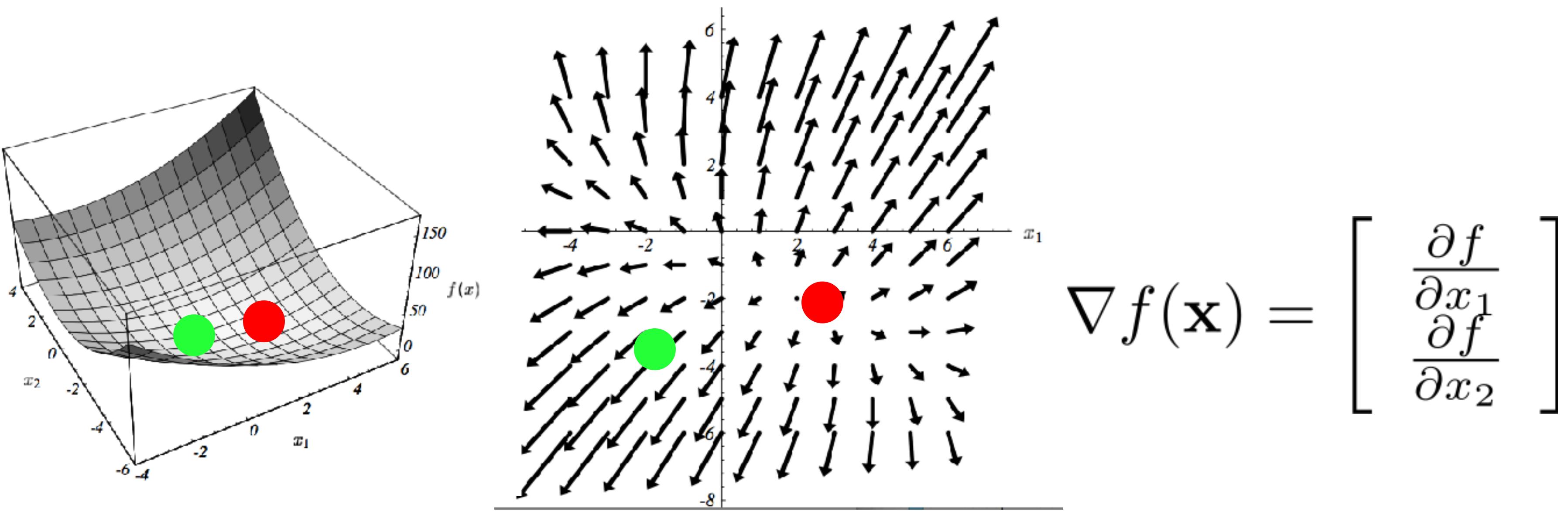
Initialize:

$$\mathbf{x}_0$$

Update:

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

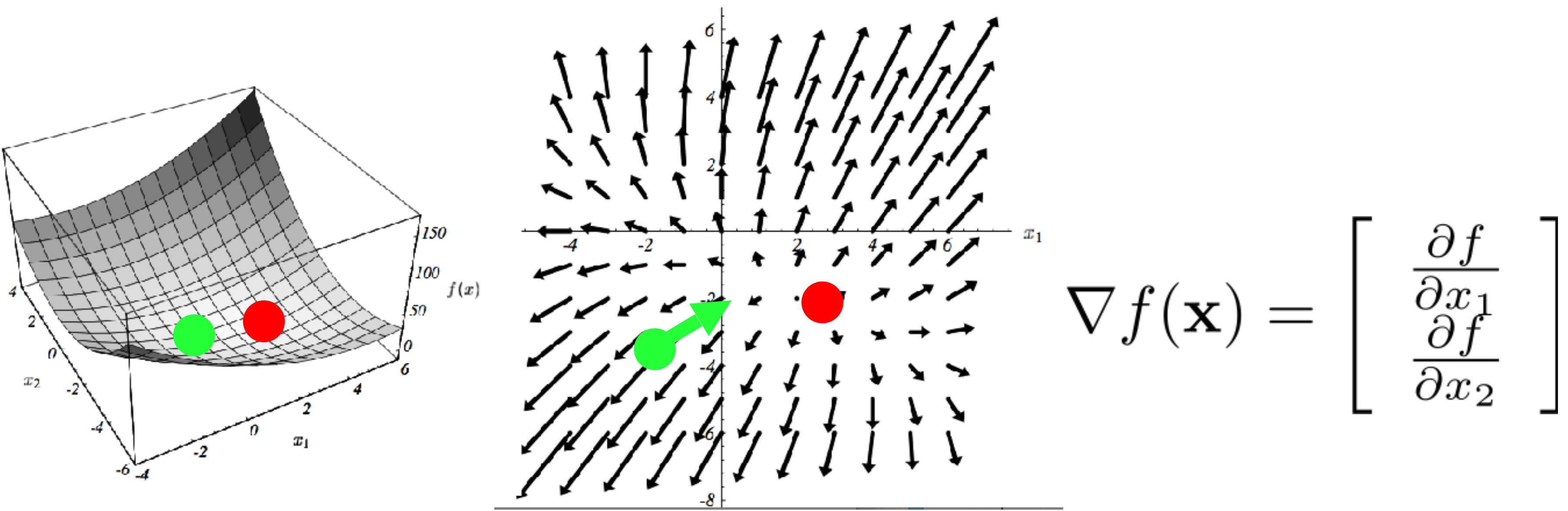
Gradient Descent Minimization



Update:

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

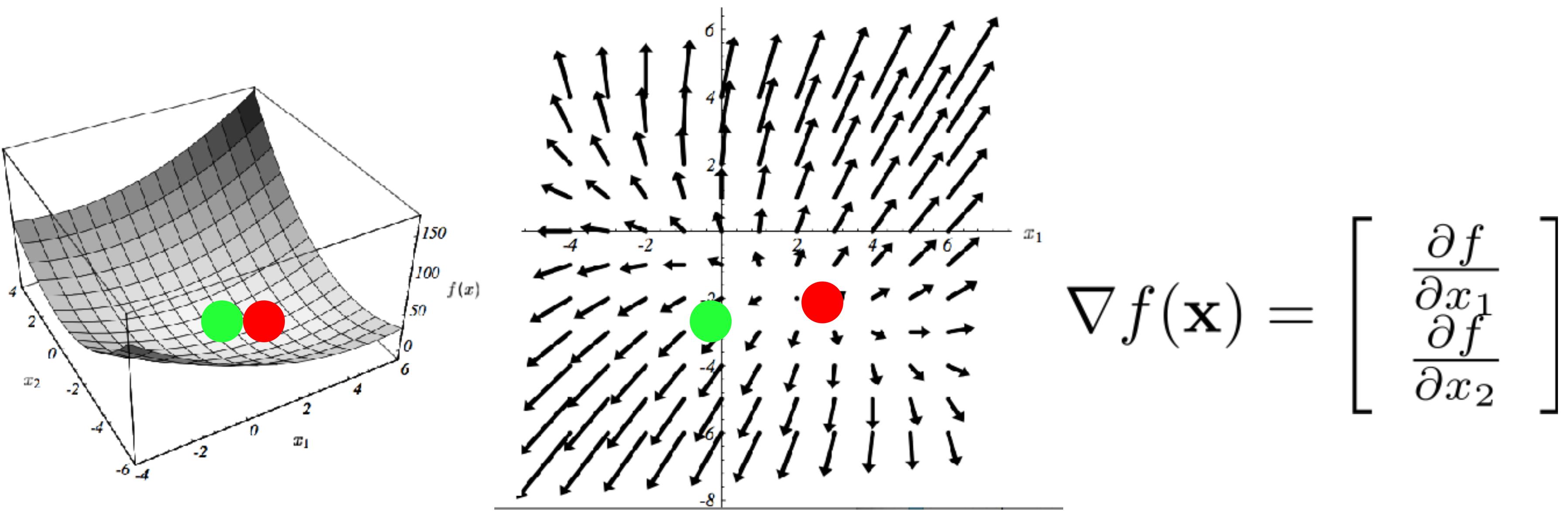
Gradient Descent Minimization



Update:

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

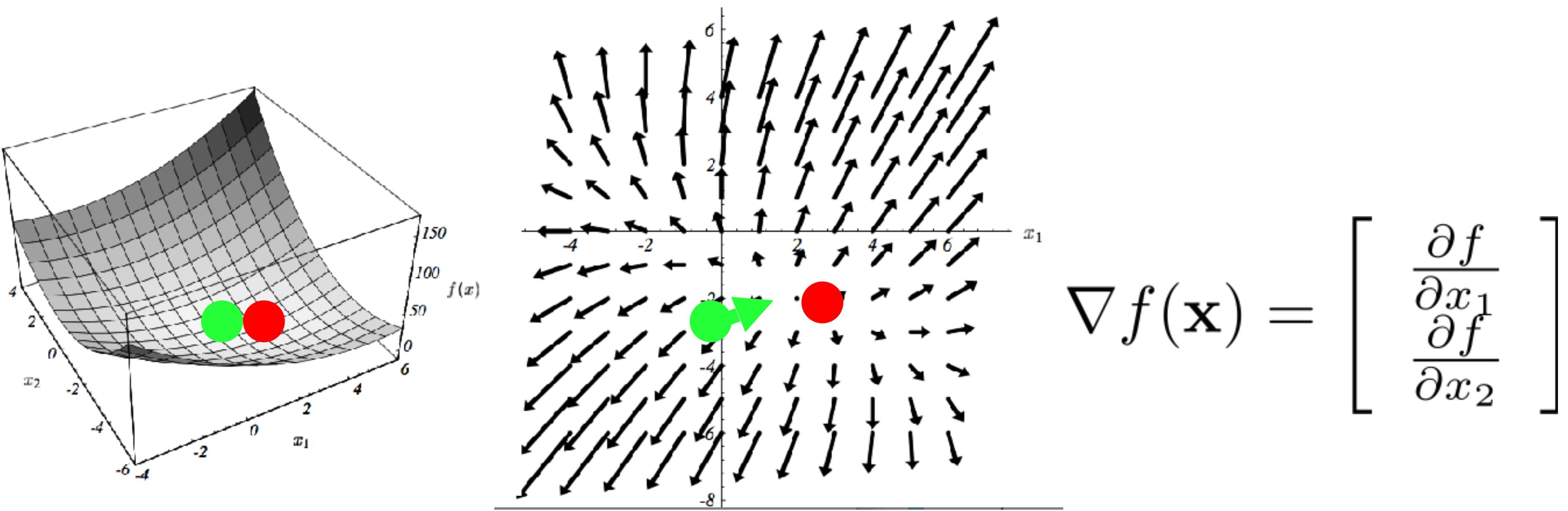
Gradient Descent Minimization



Update:

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

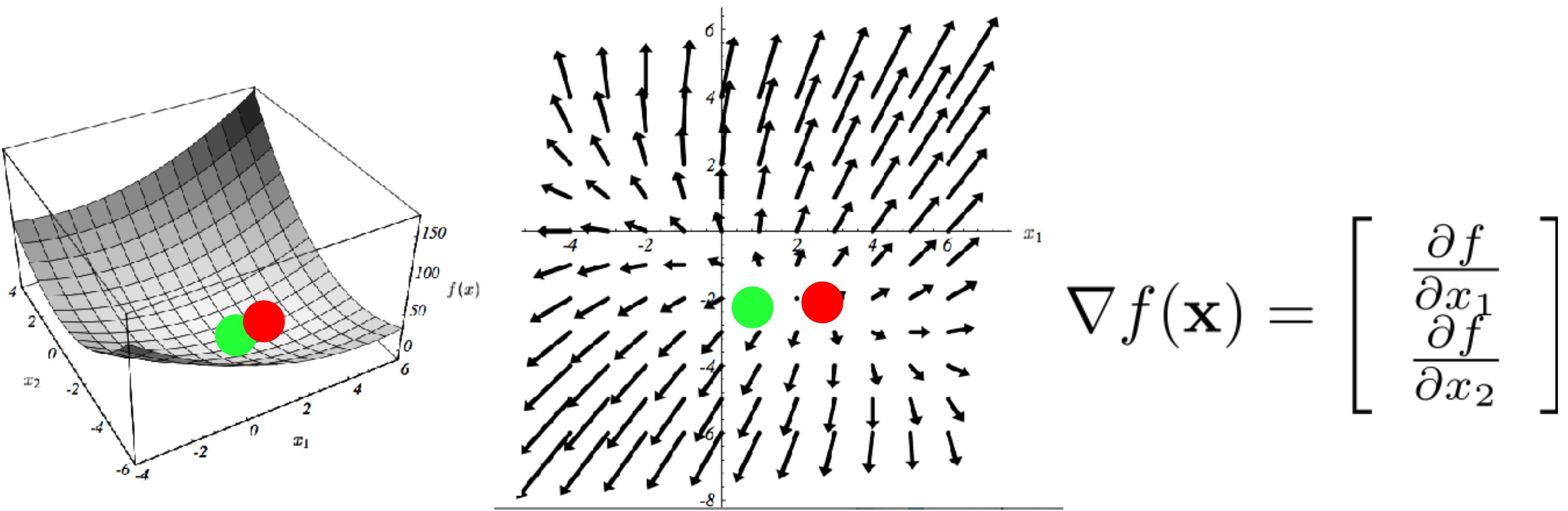
Gradient Descent Minimization



Update:

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

Gradient Descent Minimization



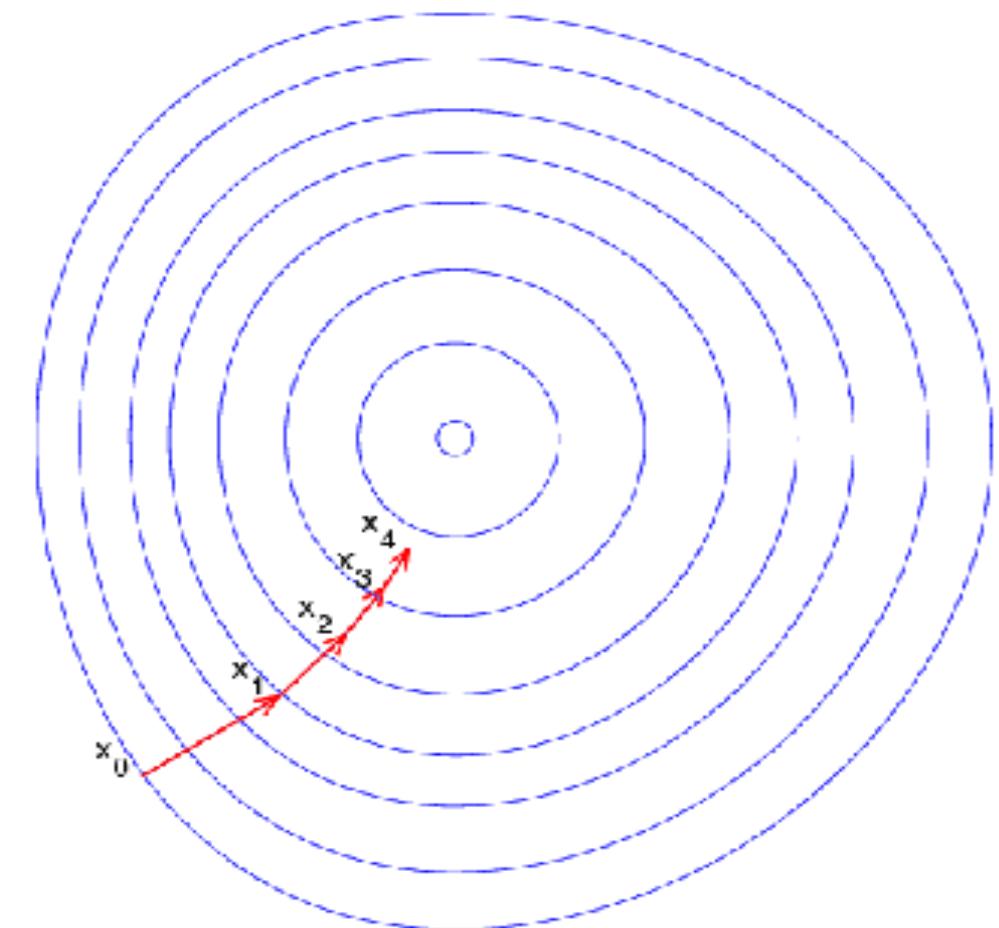
Update:

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

Gradient Descent Minimization

Initialize: \mathbf{x}_0

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

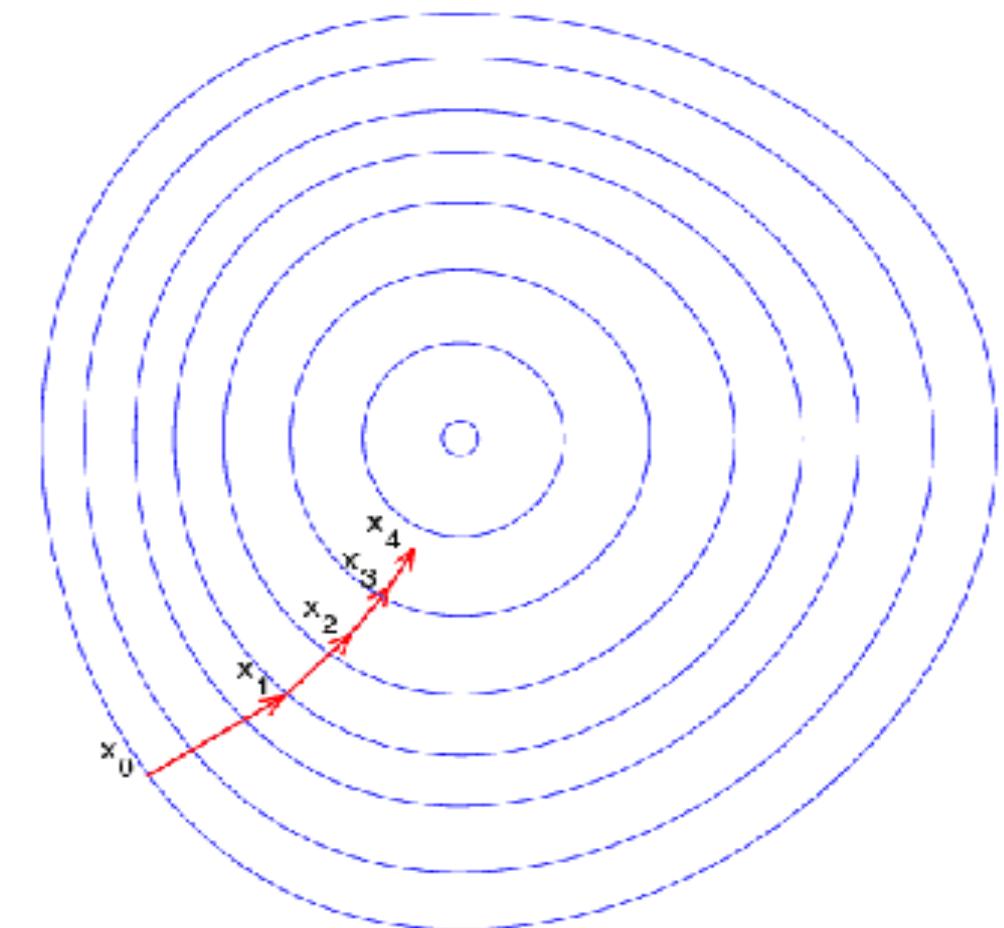


Gradient Descent Minimization

Initialize: \mathbf{x}_0

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

We can always make it converge for a convex function.

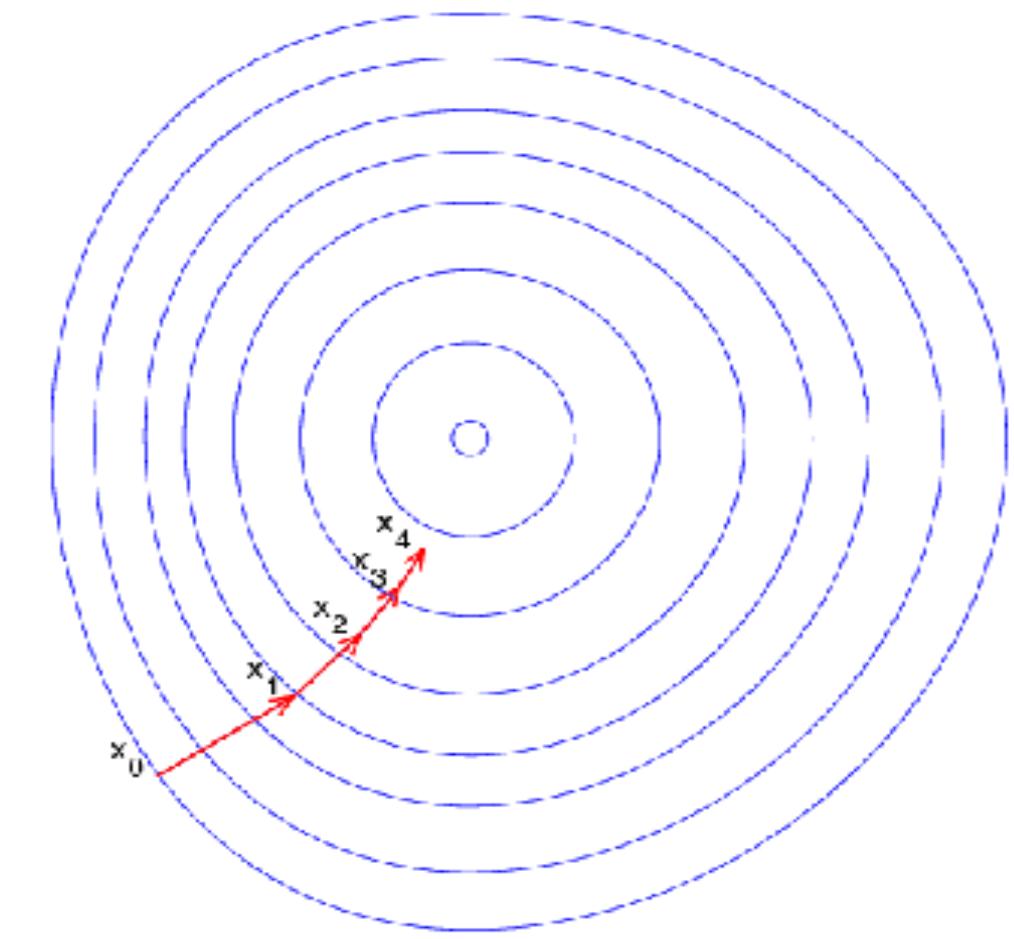
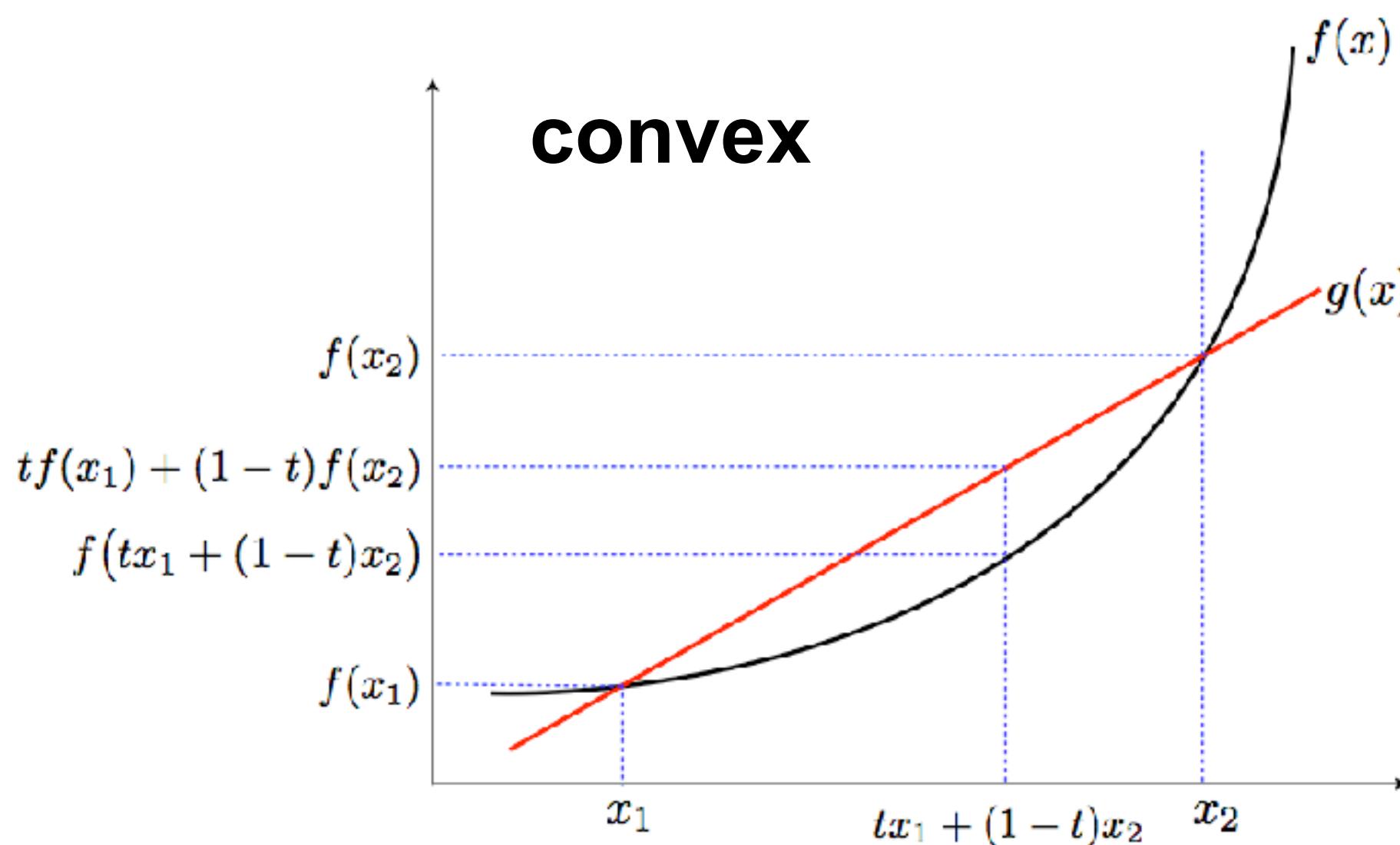


Gradient Descent Minimization

Initialize: \mathbf{x}_0

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

We can always make it converge for a convex function.

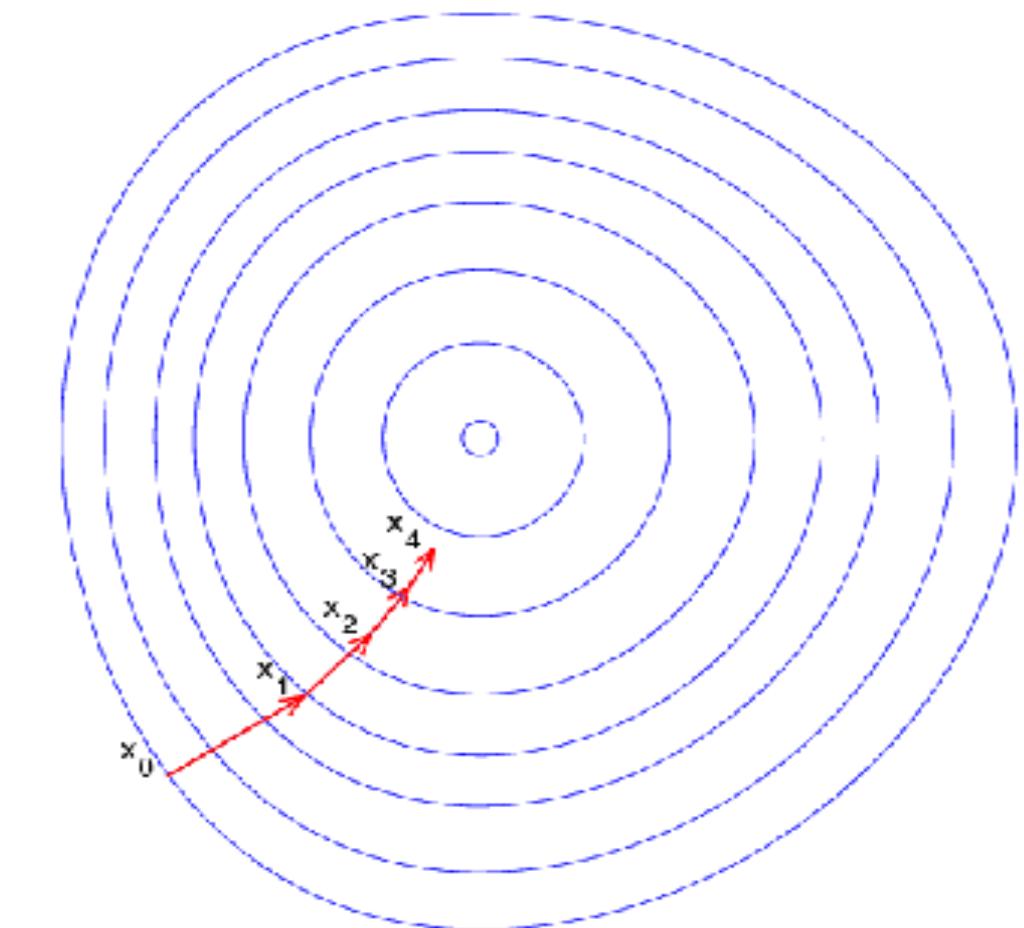
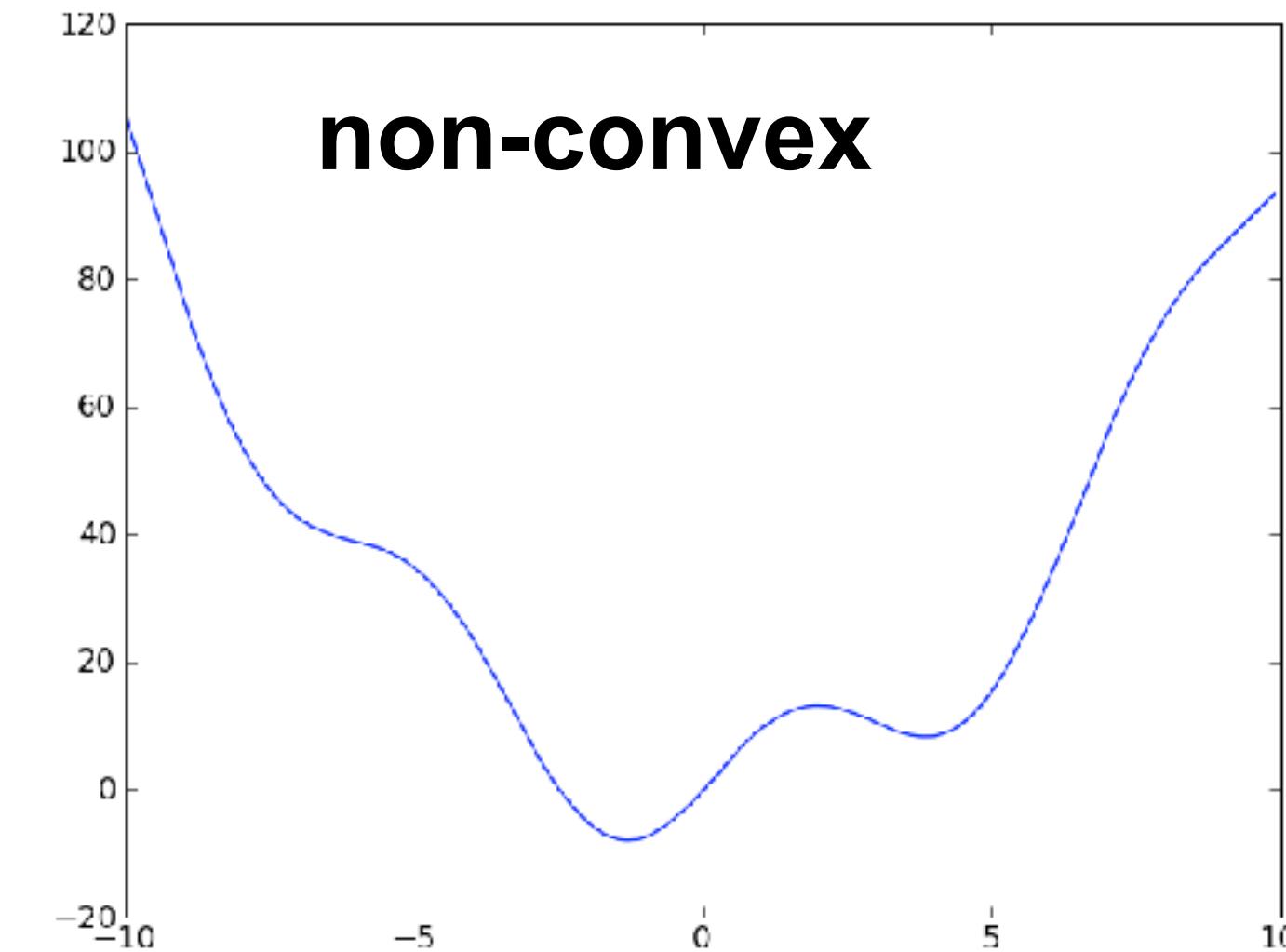
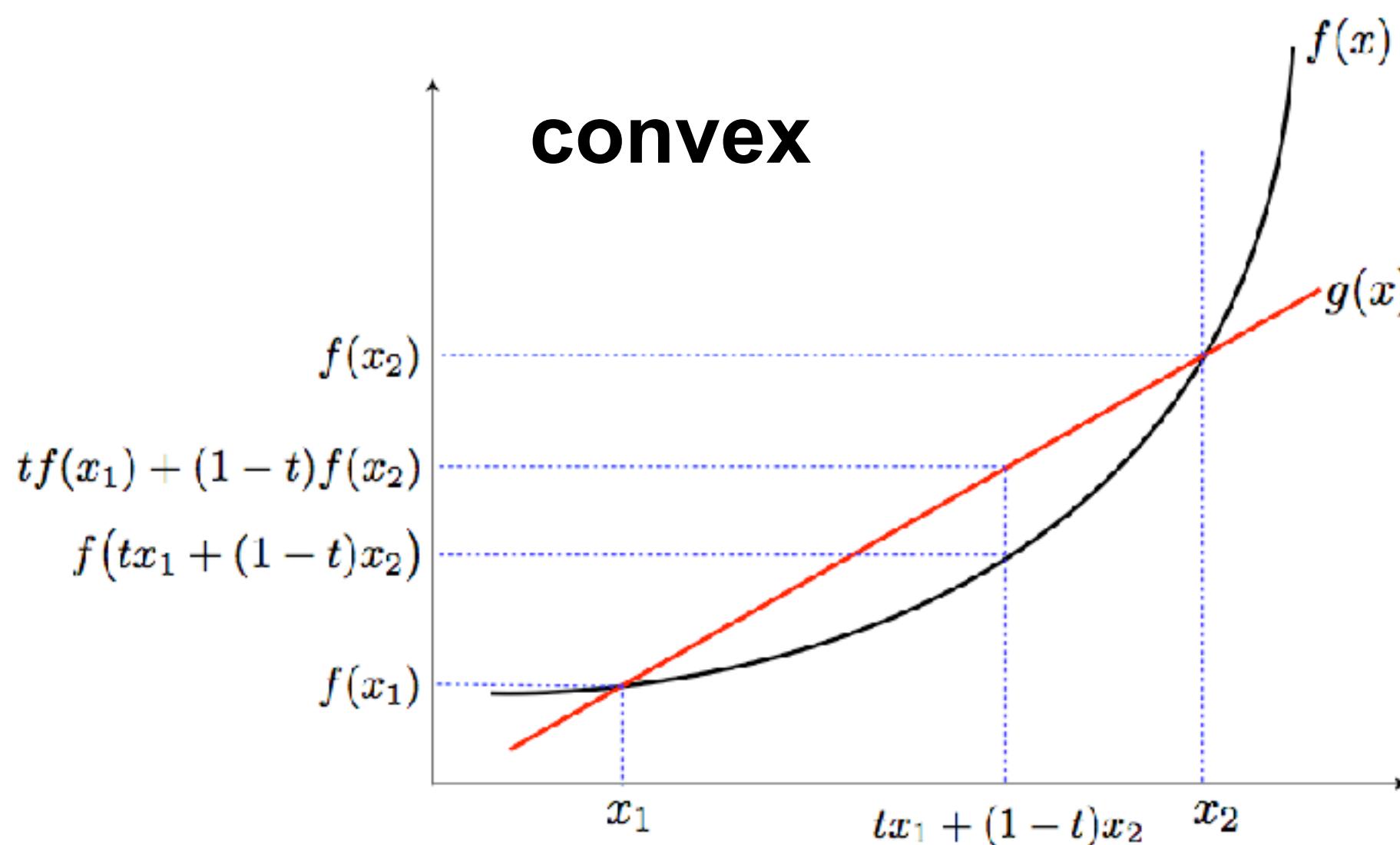


Gradient Descent Minimization

Initialize: \mathbf{x}_0

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

We can always make it converge for a convex function.



XOR Problem

XOR Problem

XOR Problem

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

XOR Problem

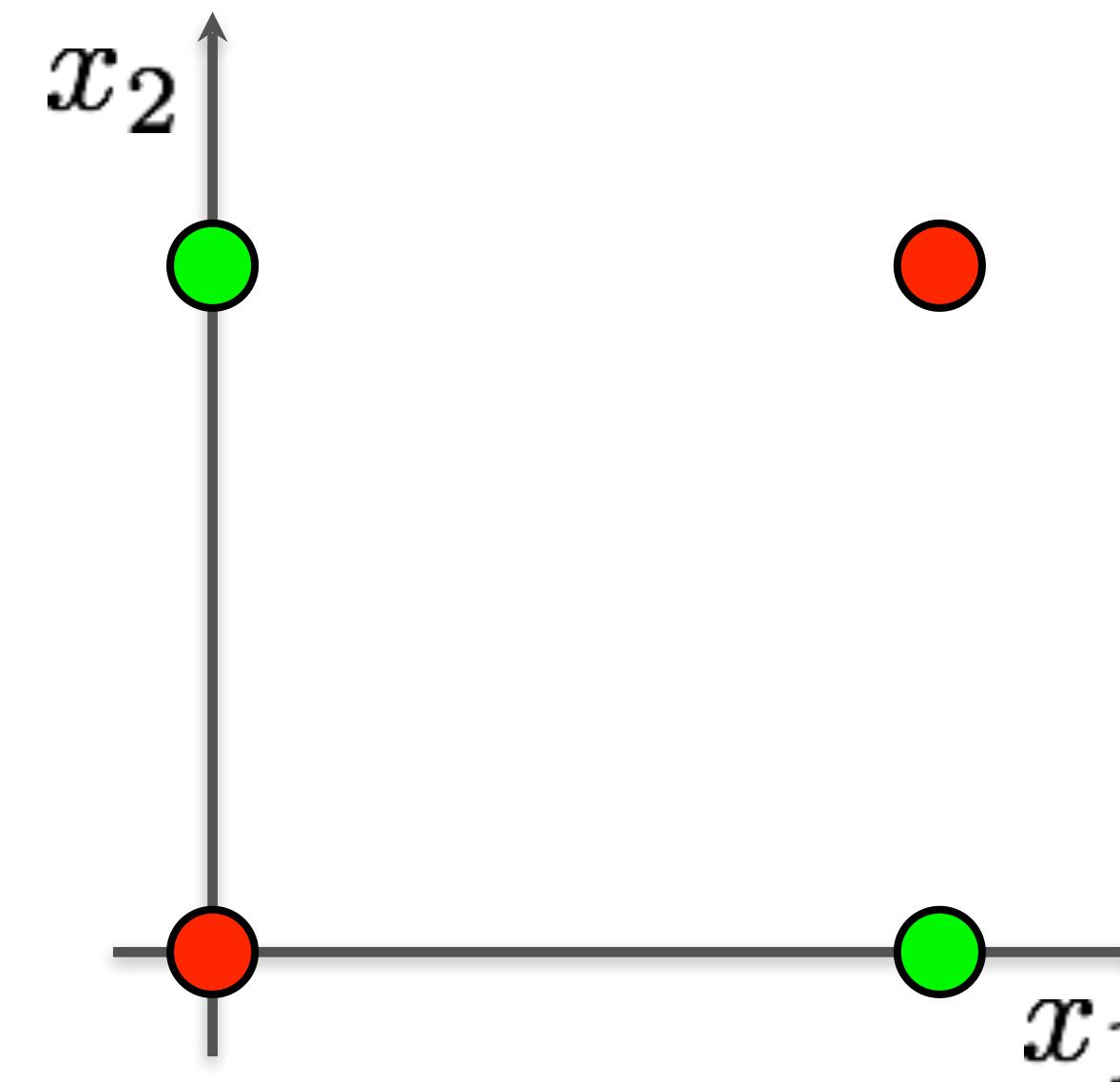
$$y = f(x_1, x_2)$$

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

XOR Problem

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

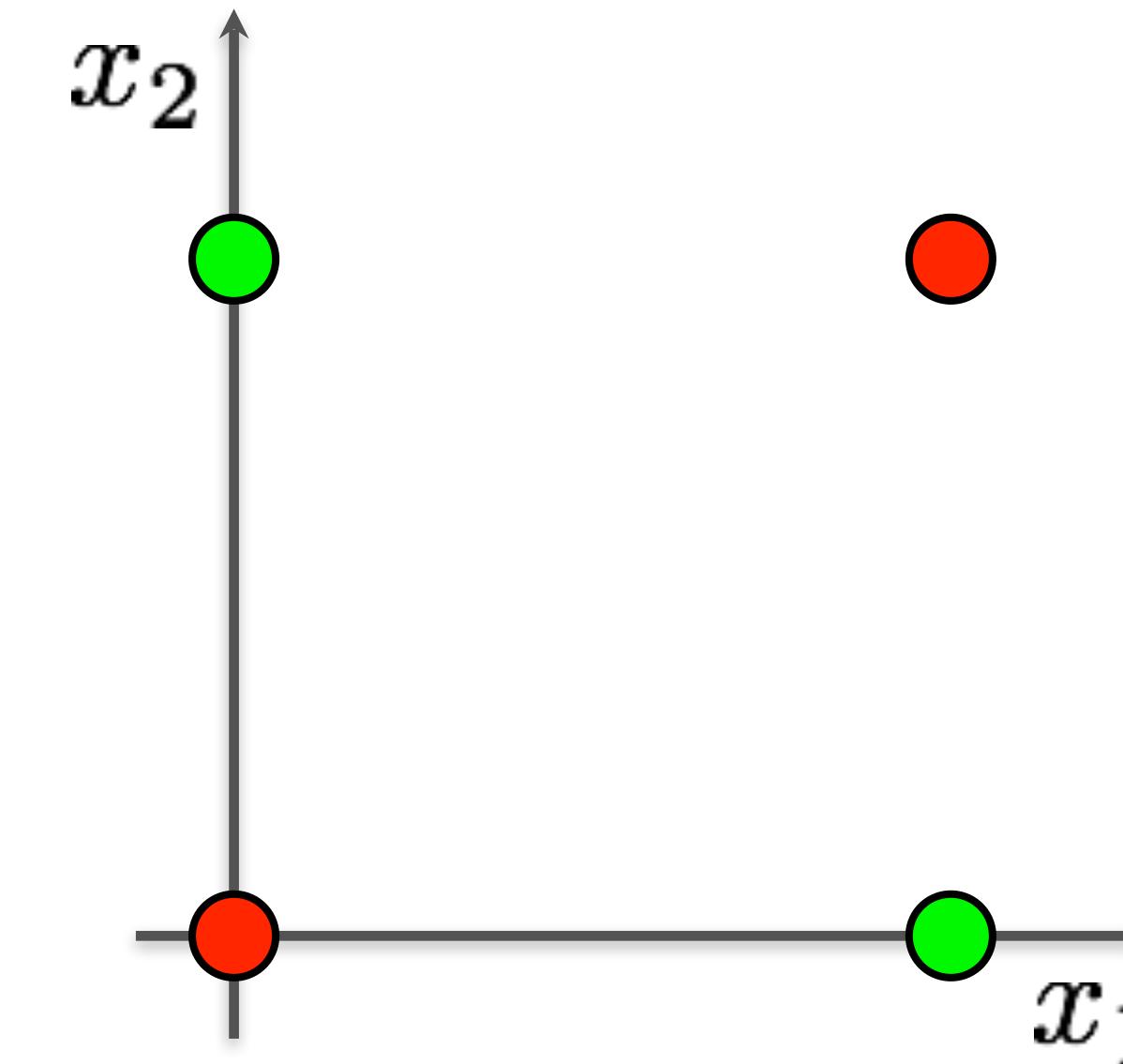
$$y = f(x_1, x_2)$$



XOR Problem

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

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

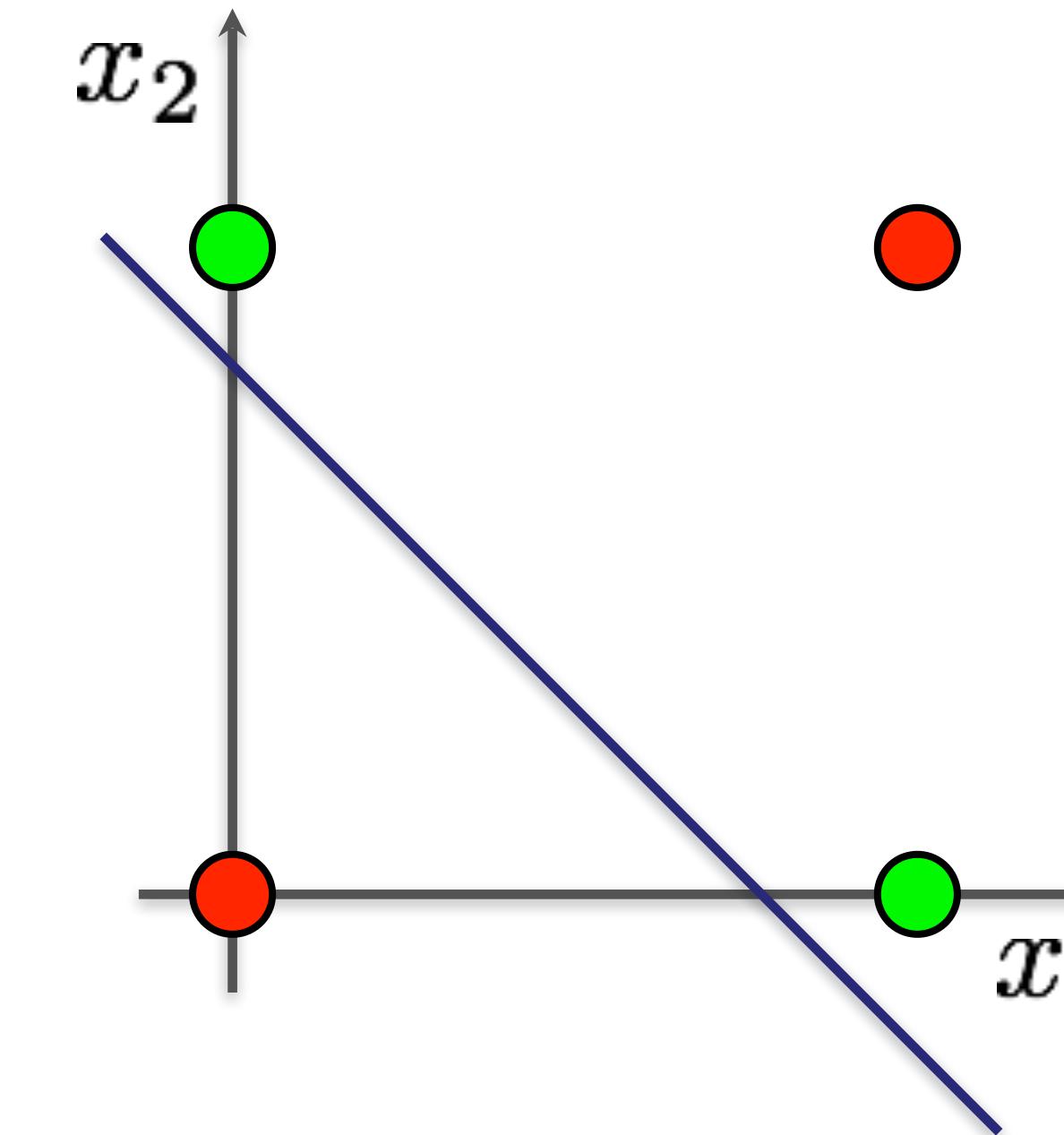


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

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

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

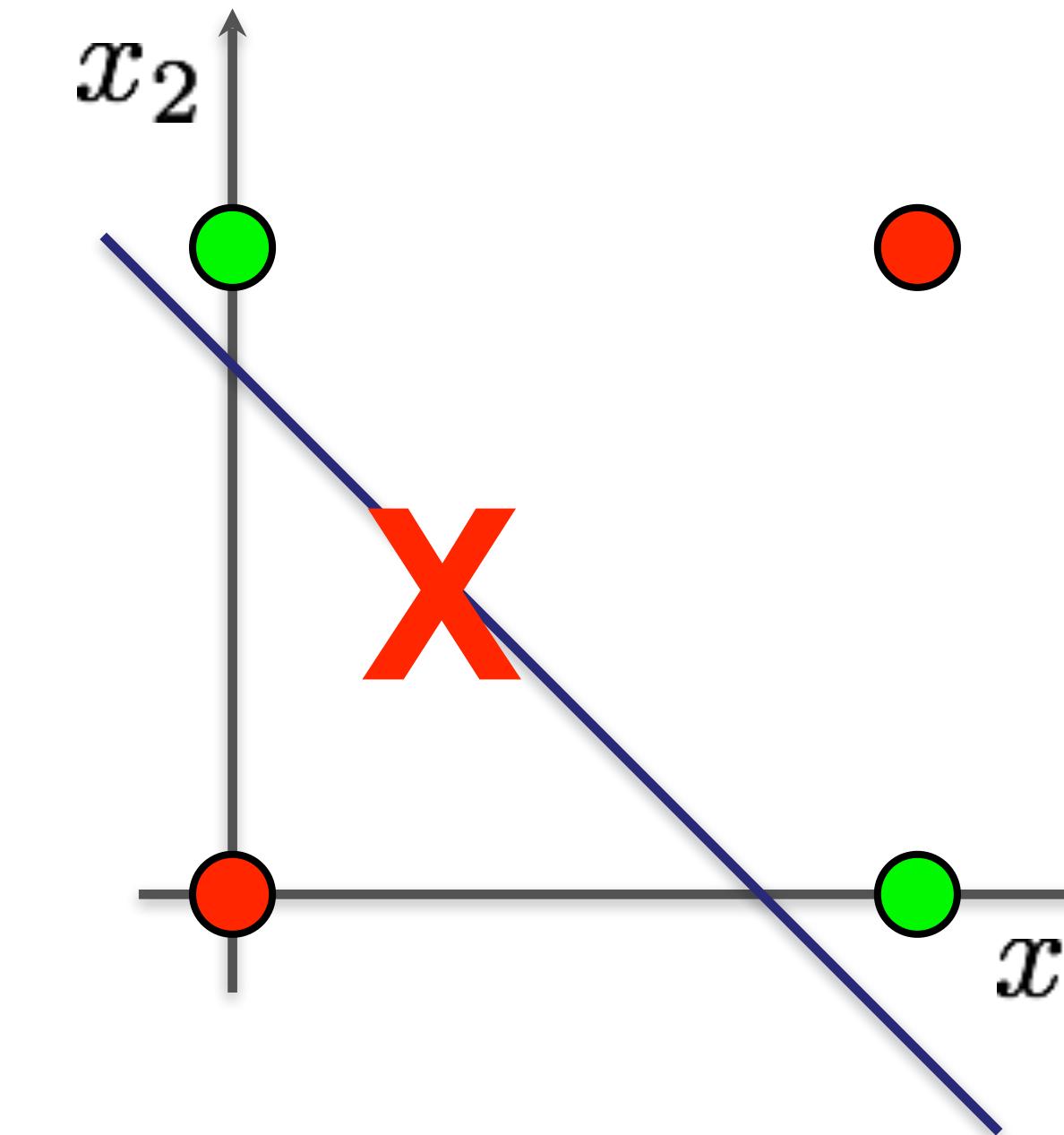


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

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

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

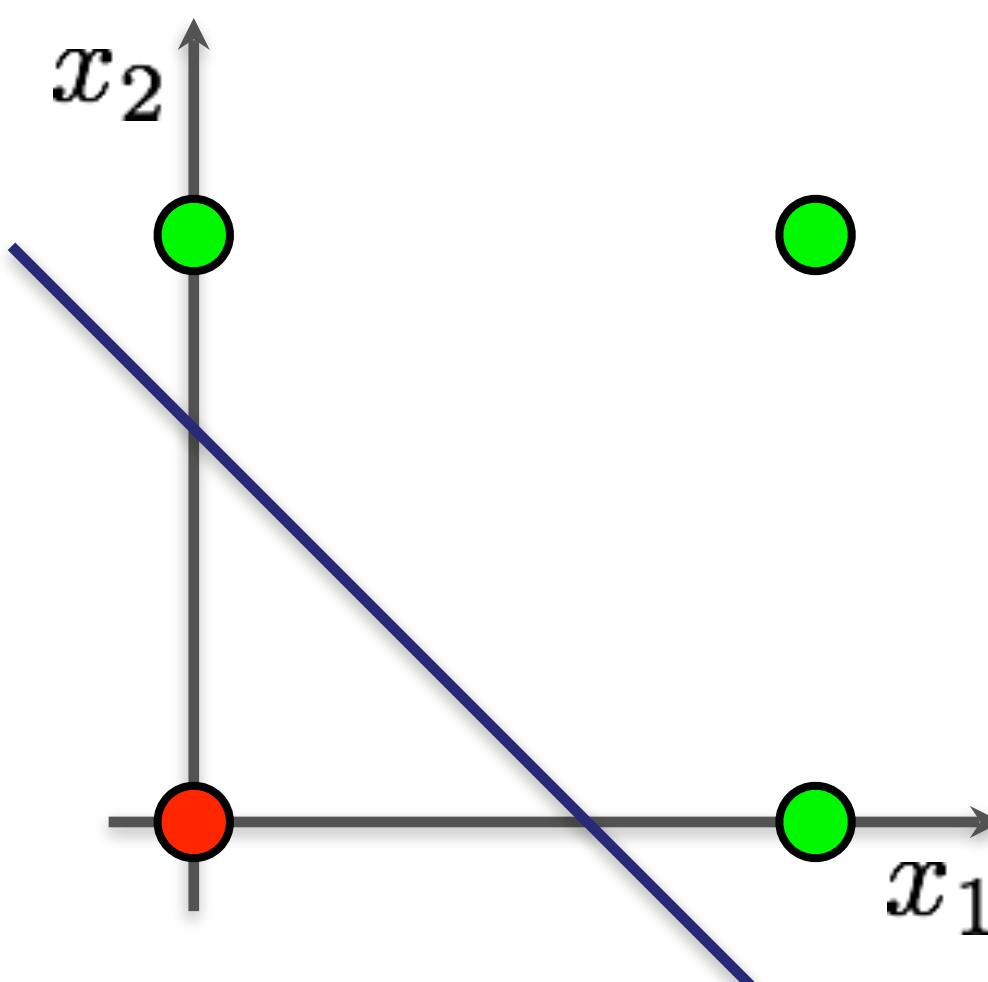


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

XOR Problem

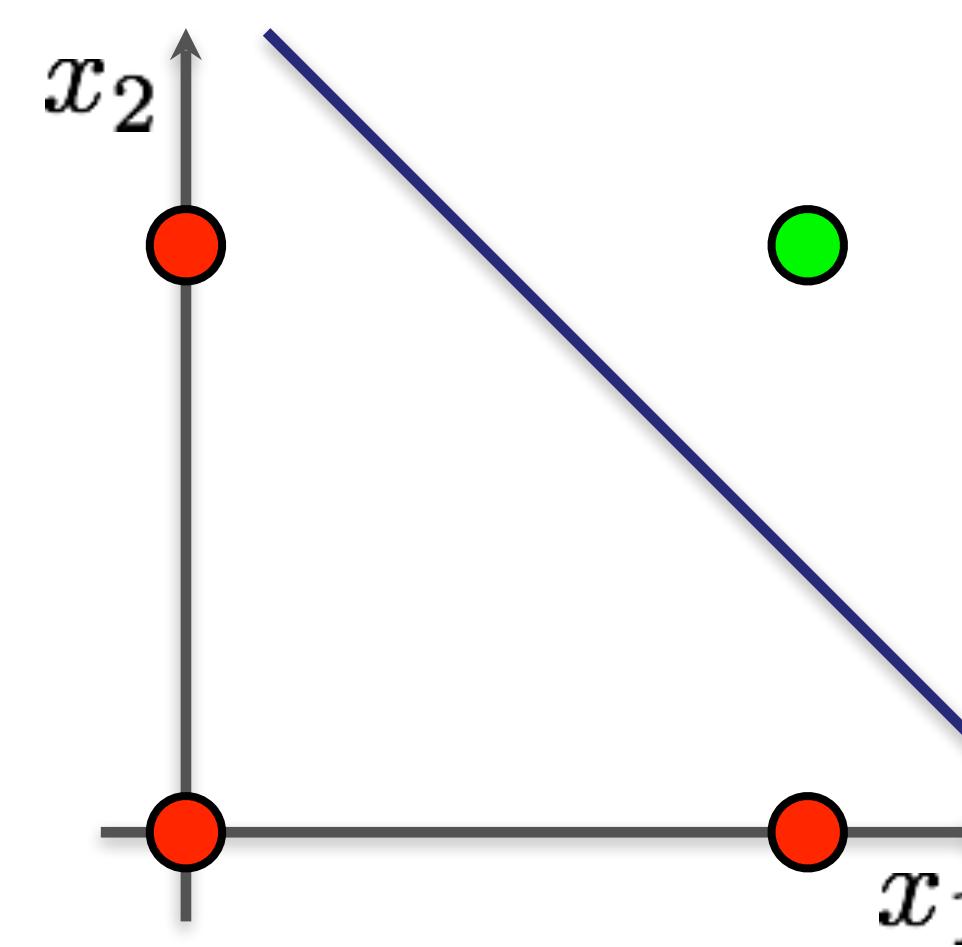
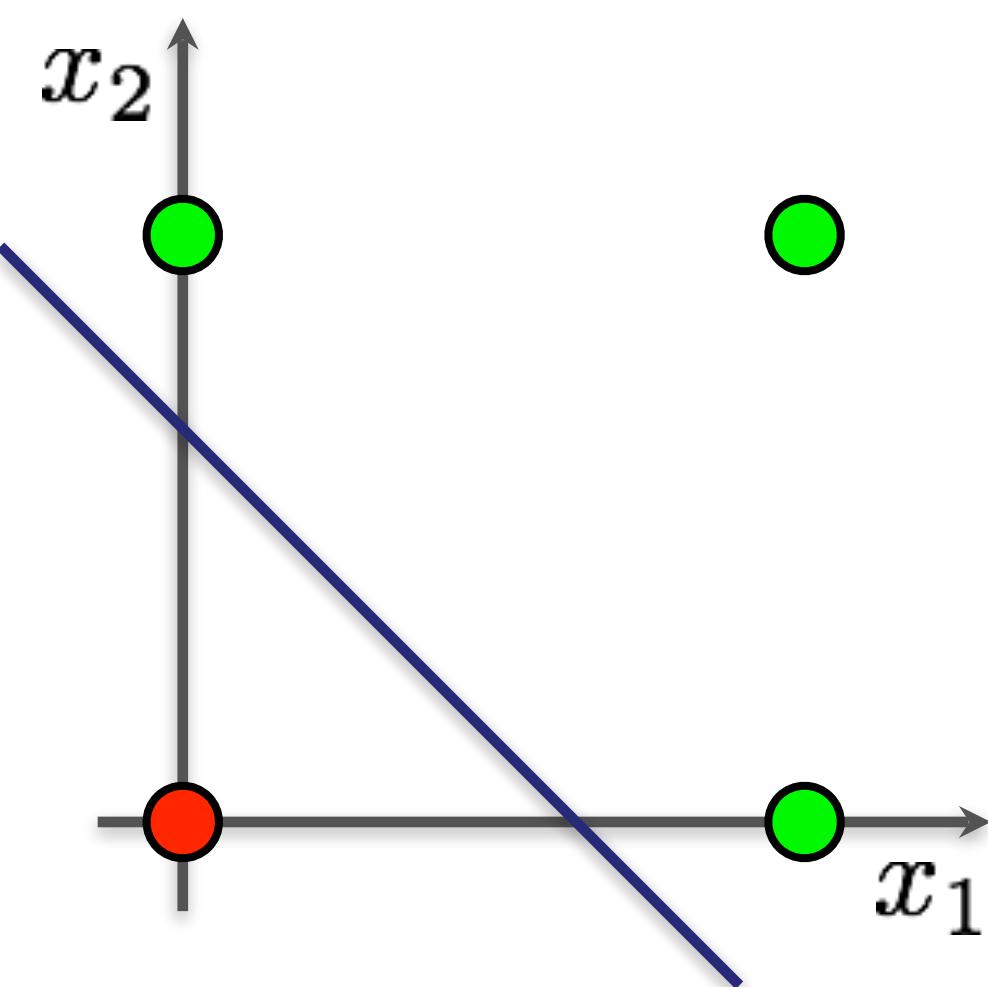
x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1



XOR Problem

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

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

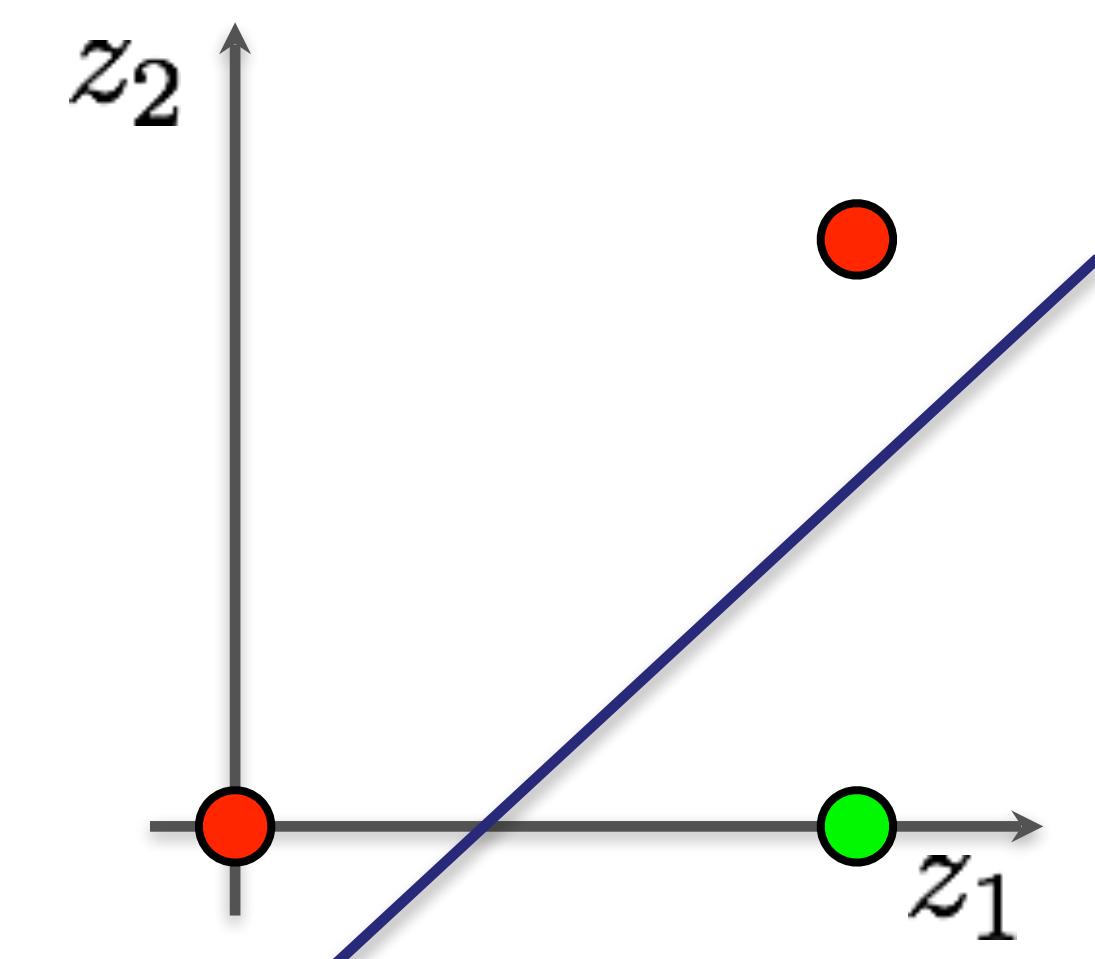
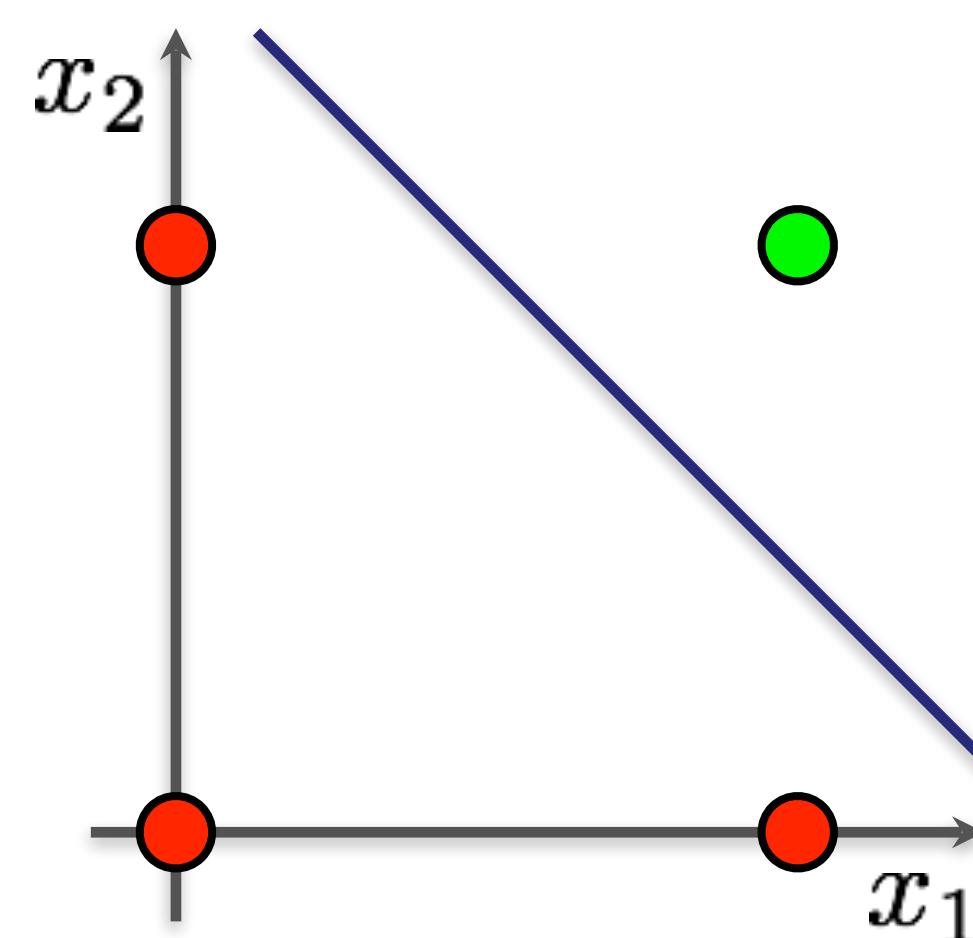
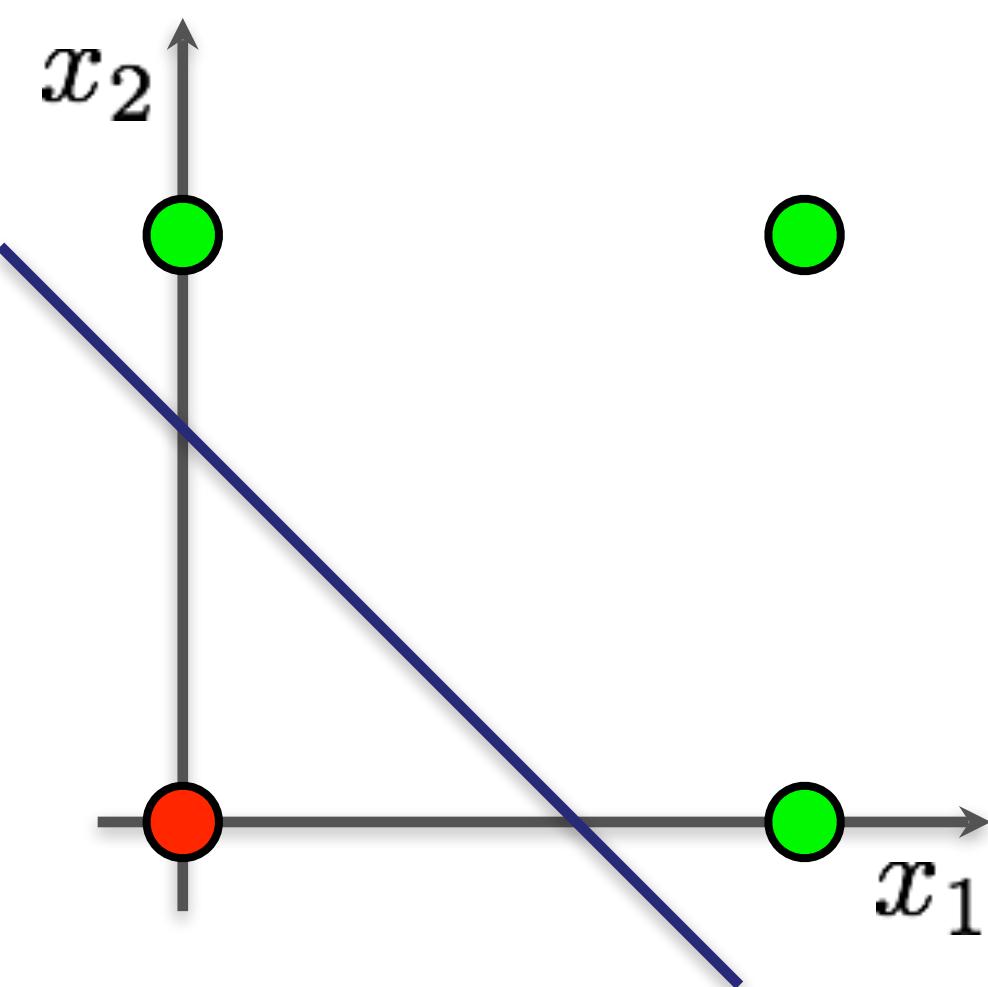


XOR Problem

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

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

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



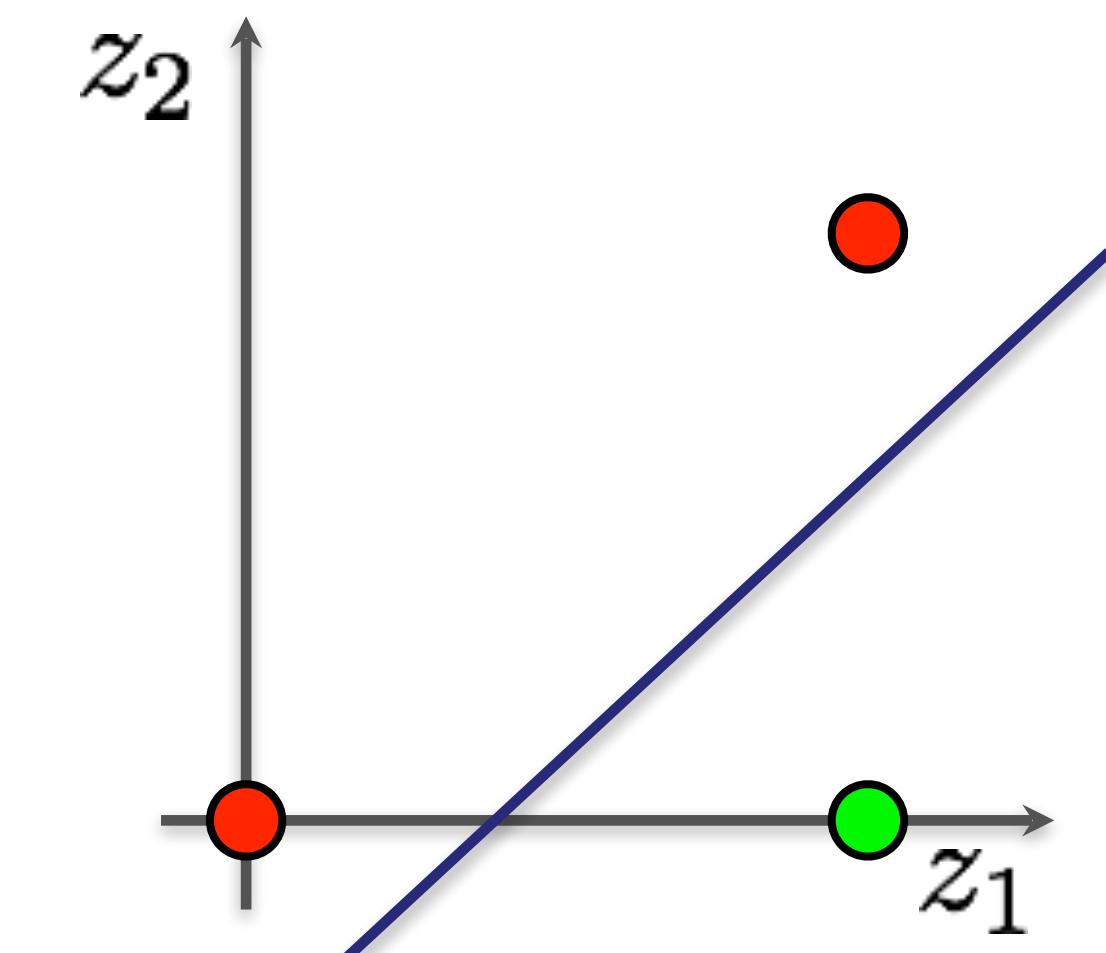
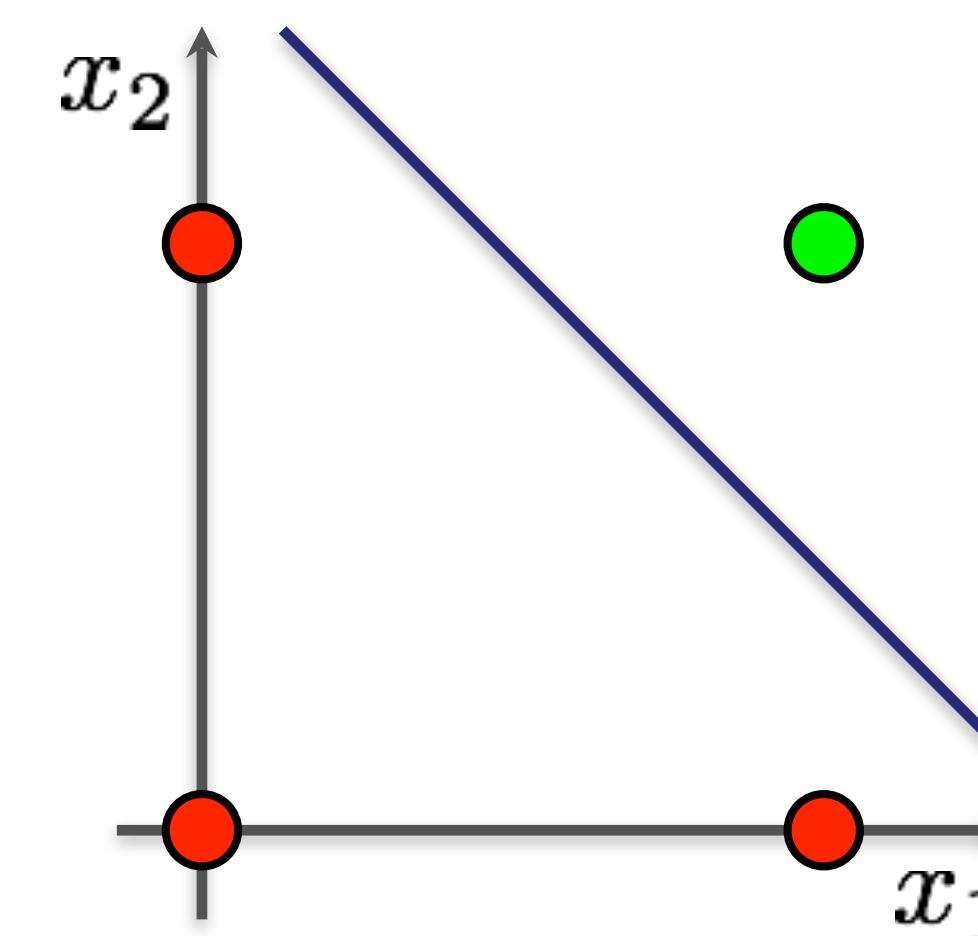
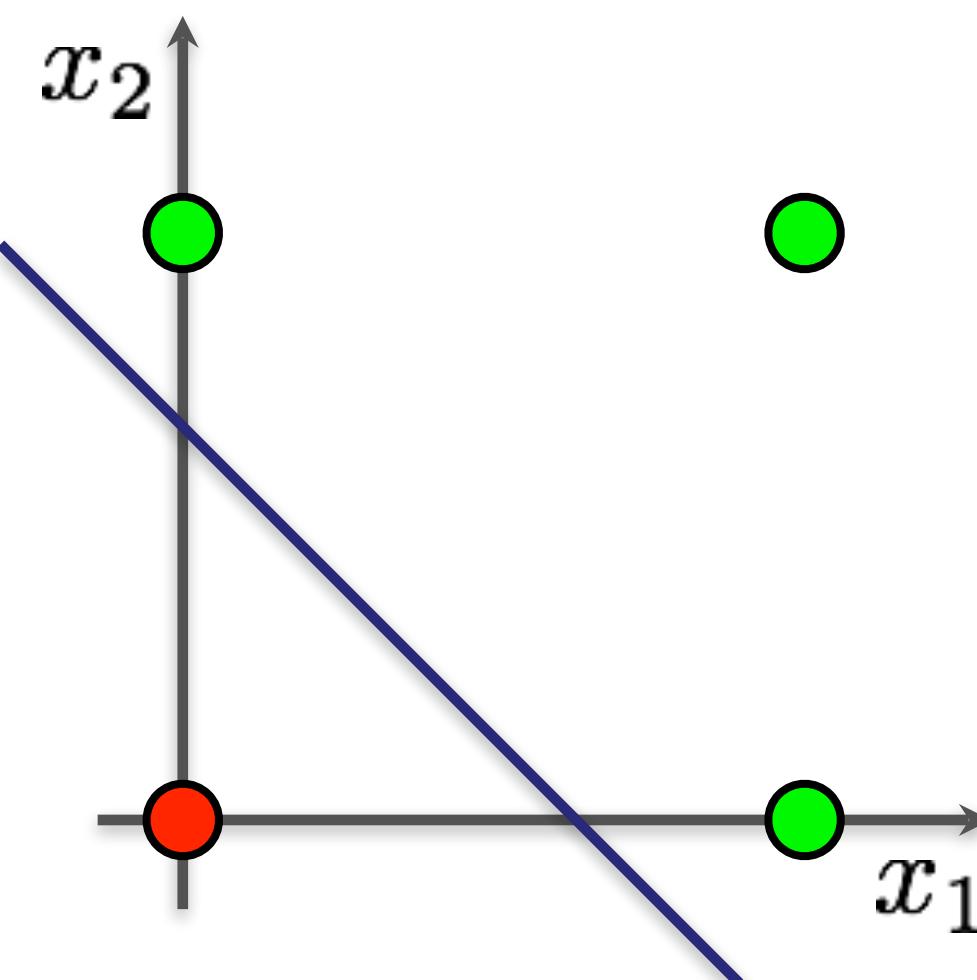
XOR Problem

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

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

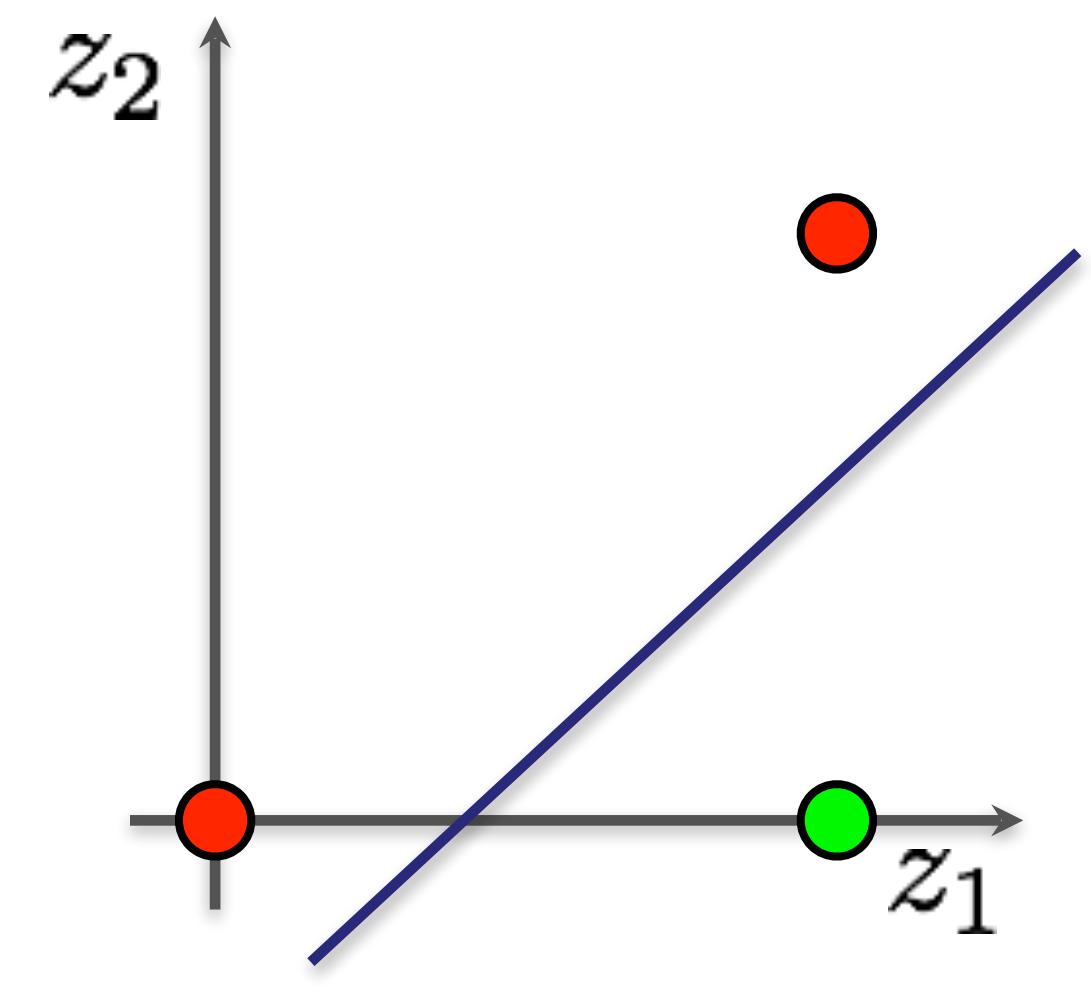
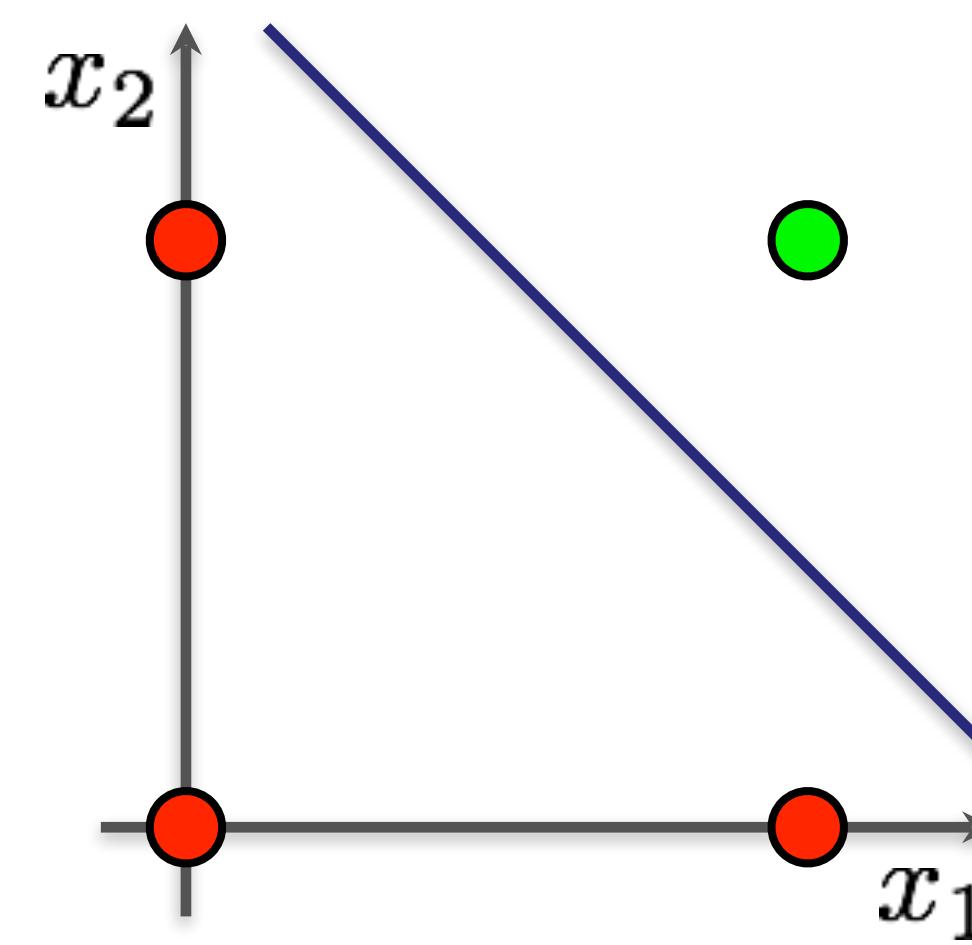
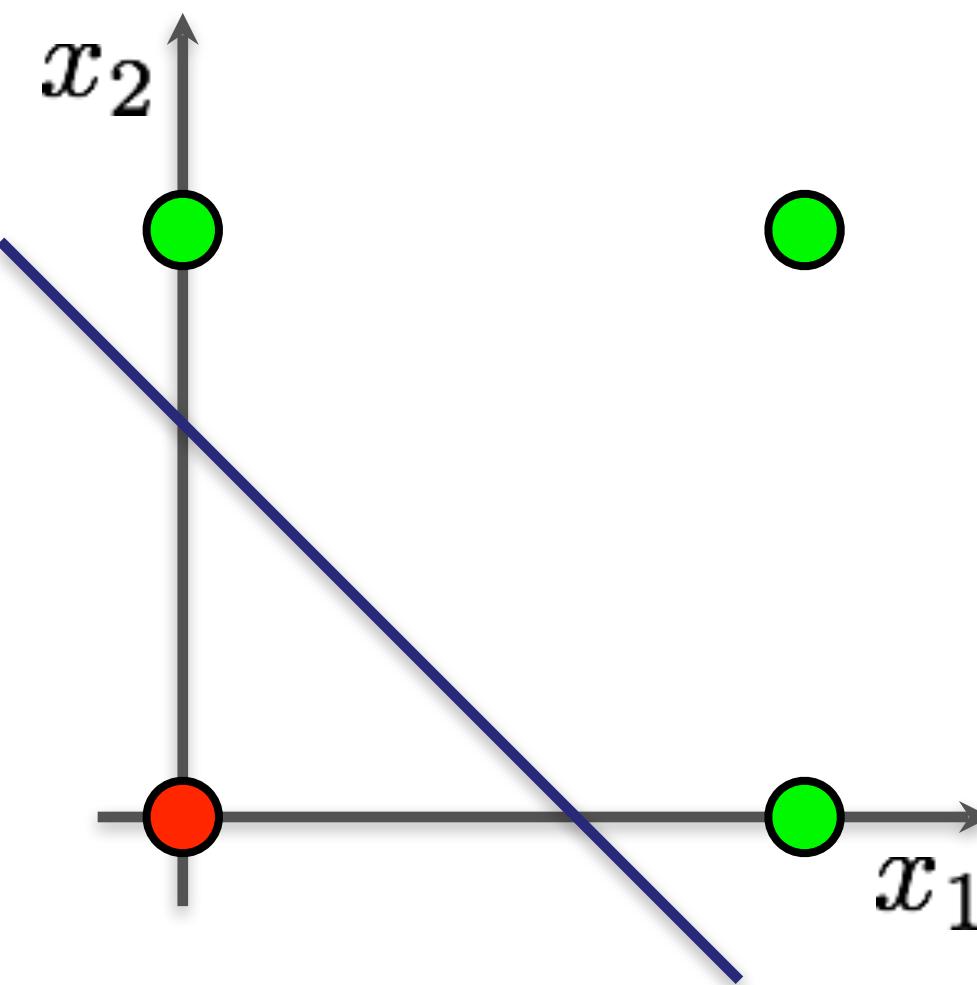
x_1	x_2	z_2
0	0	0
0	1	0
1	0	0
1	1	1

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



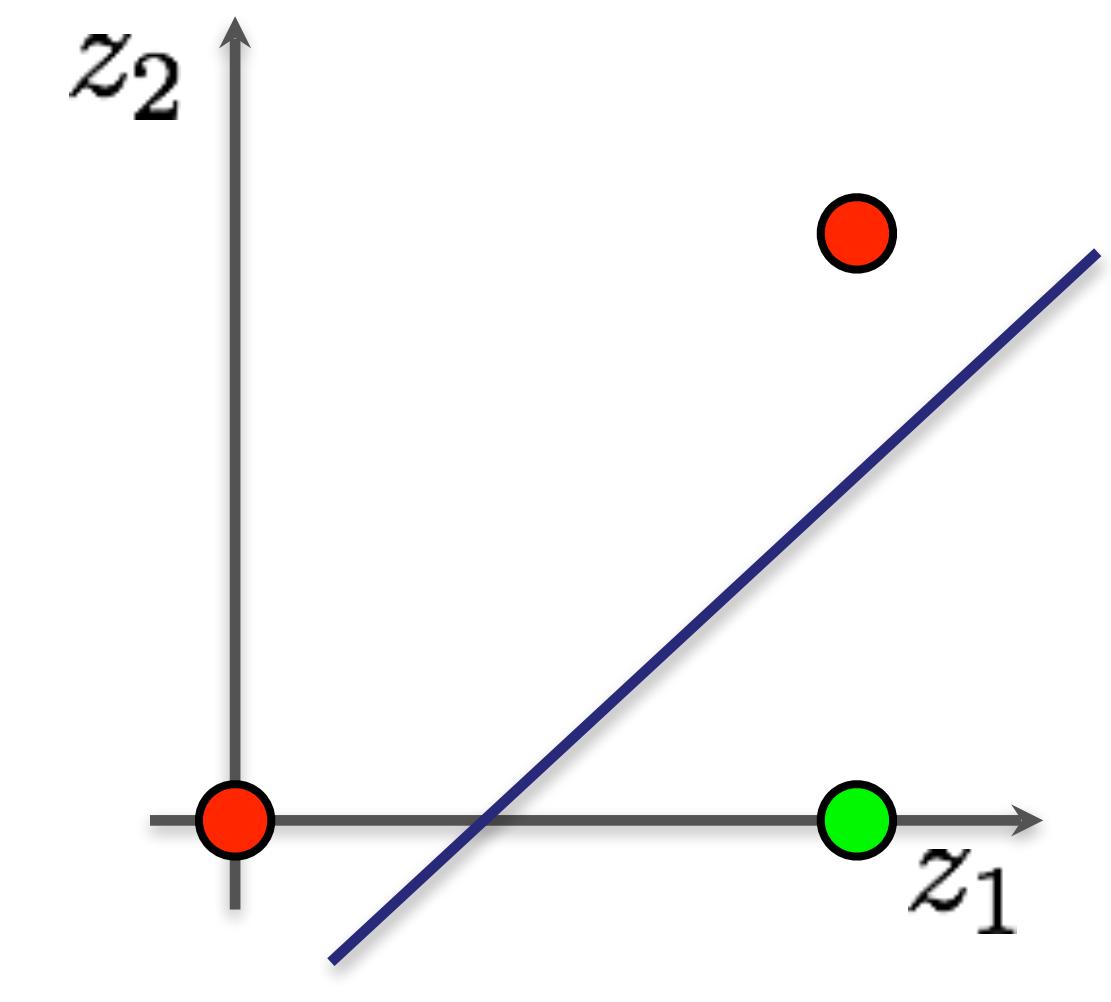
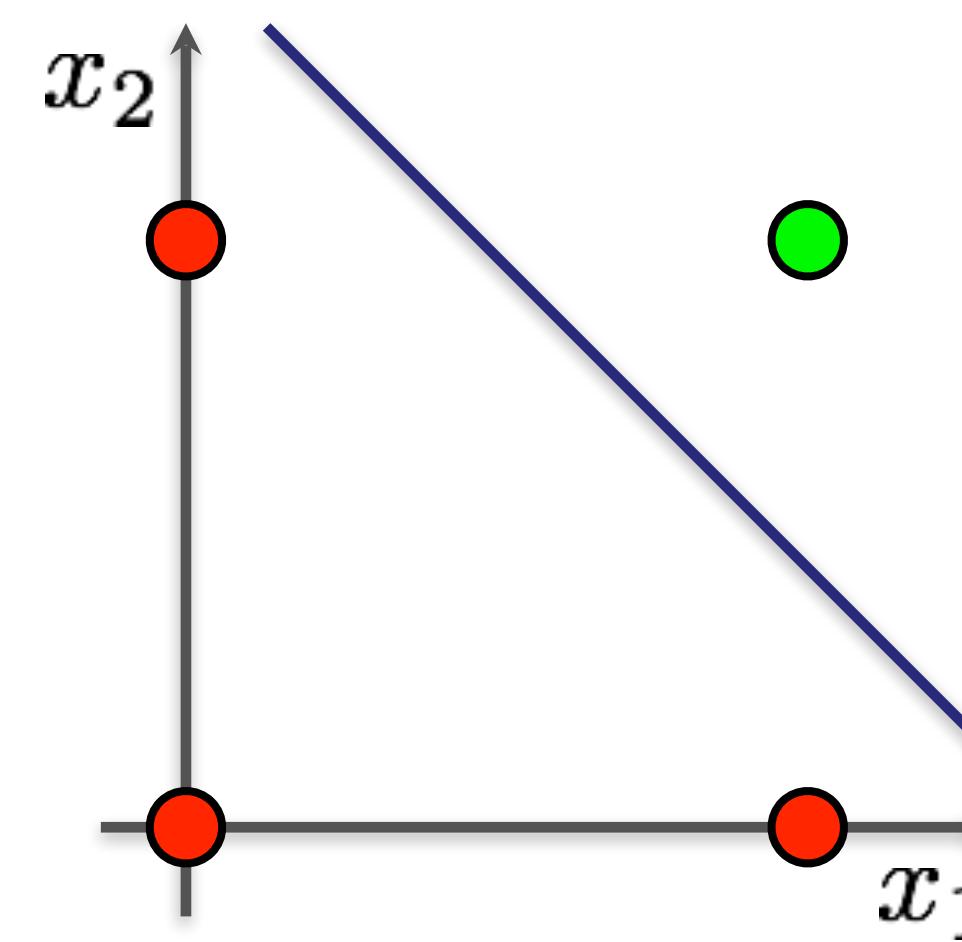
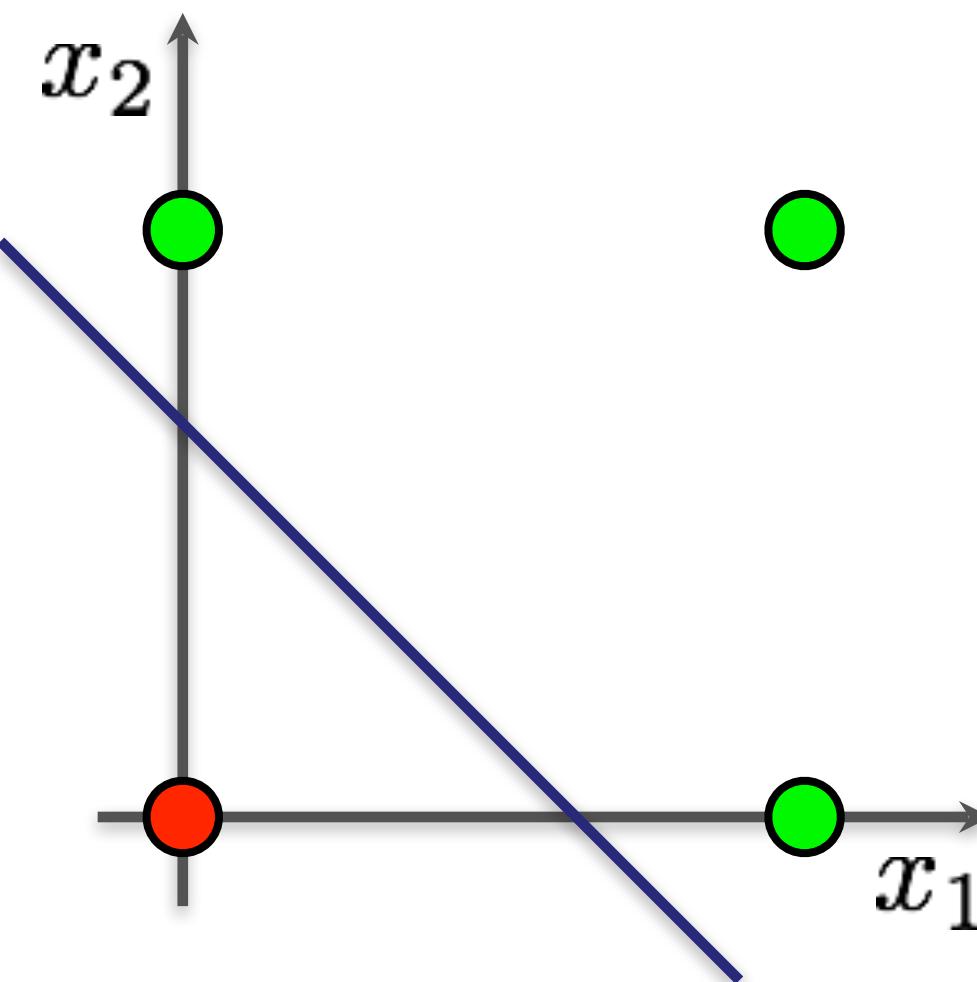
XOR Problem

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



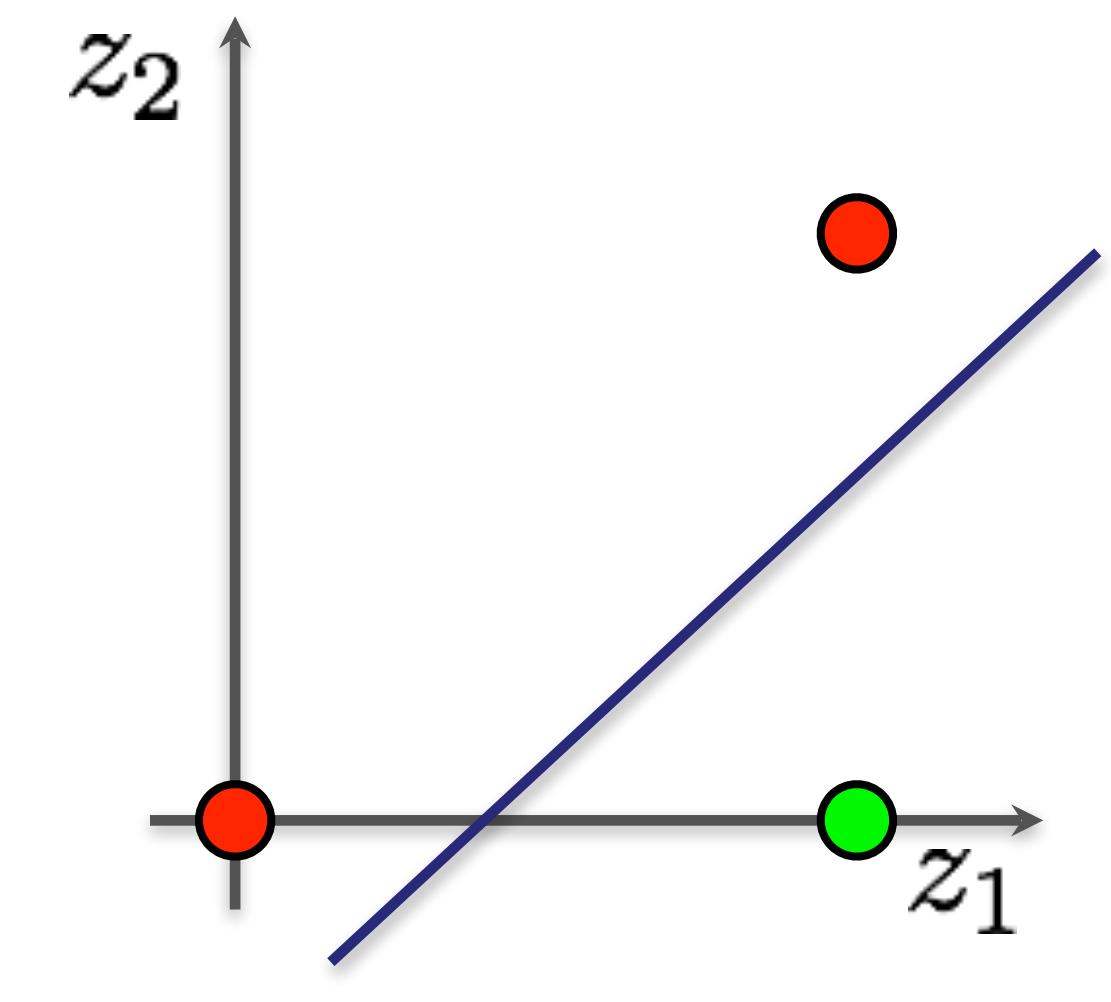
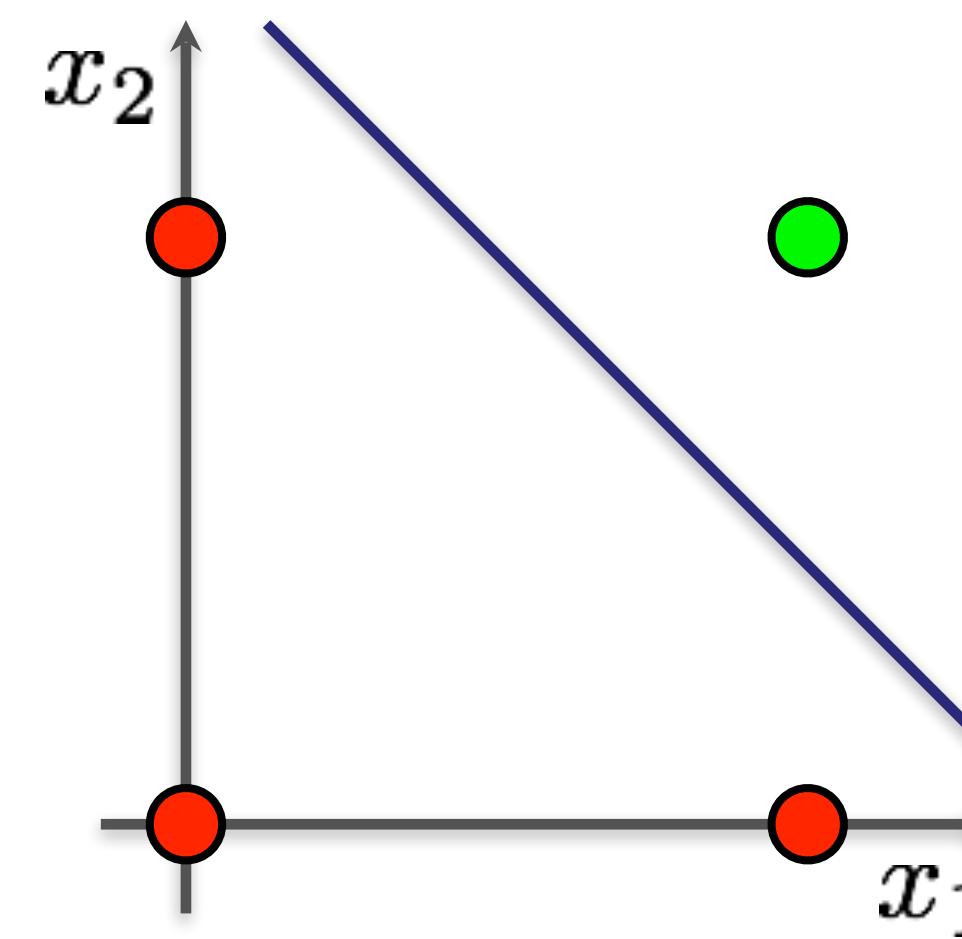
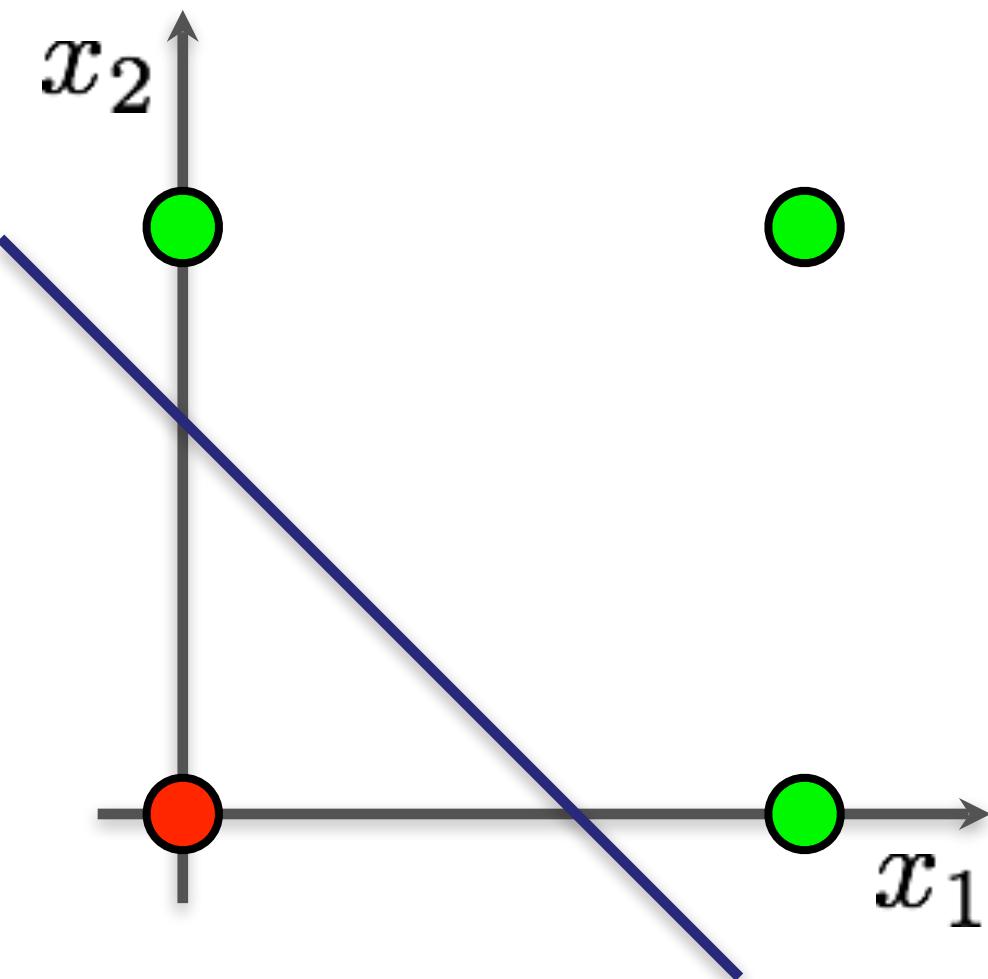
XOR Problem

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



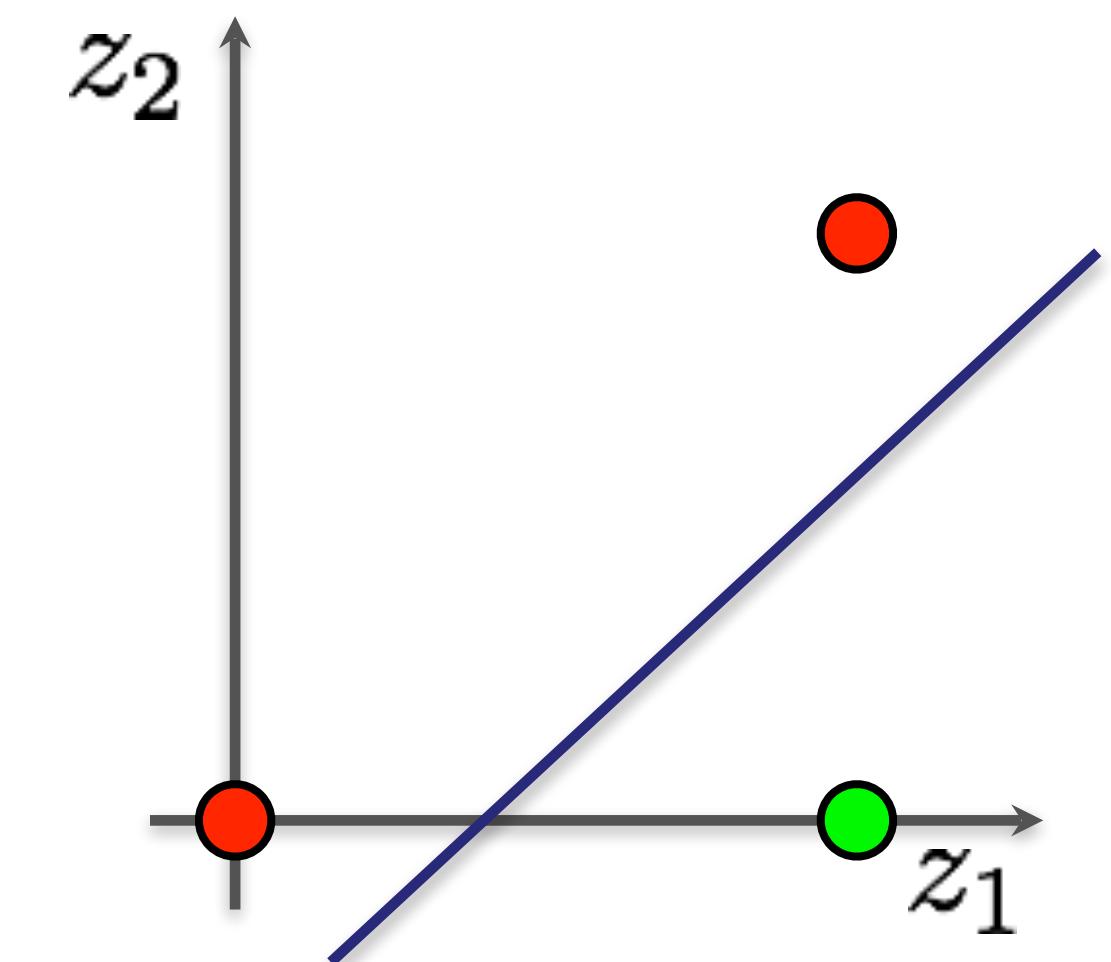
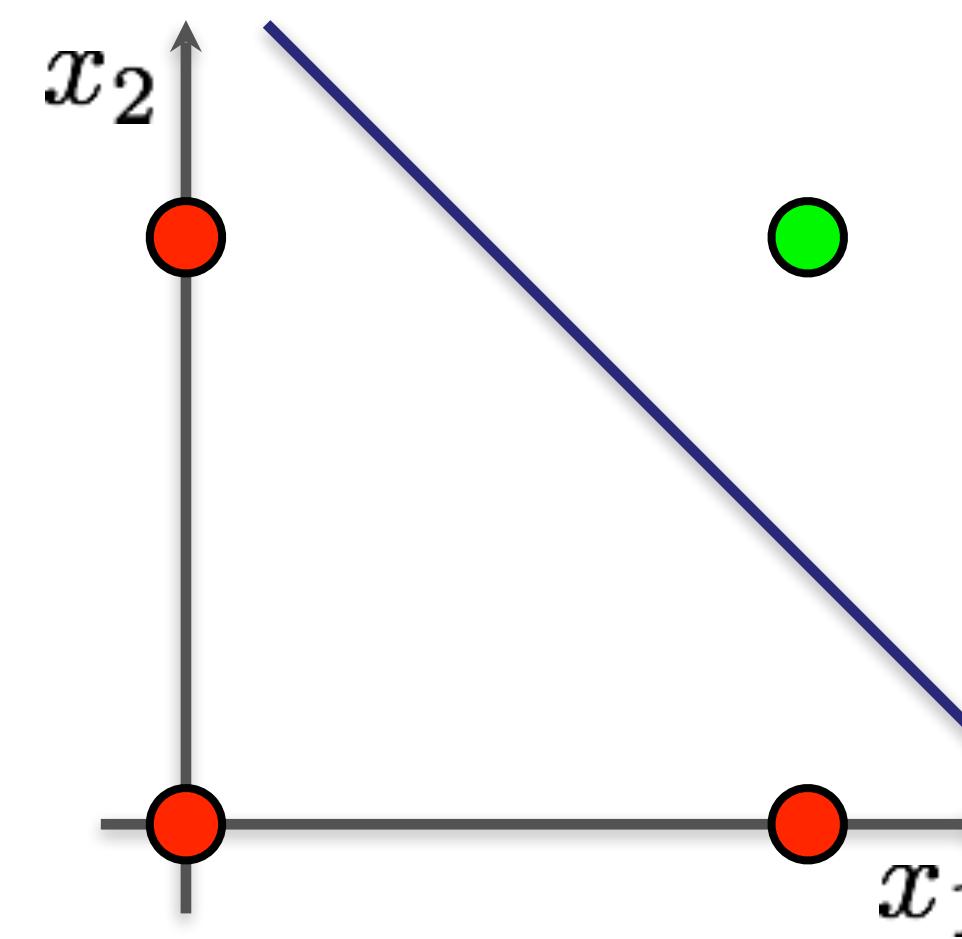
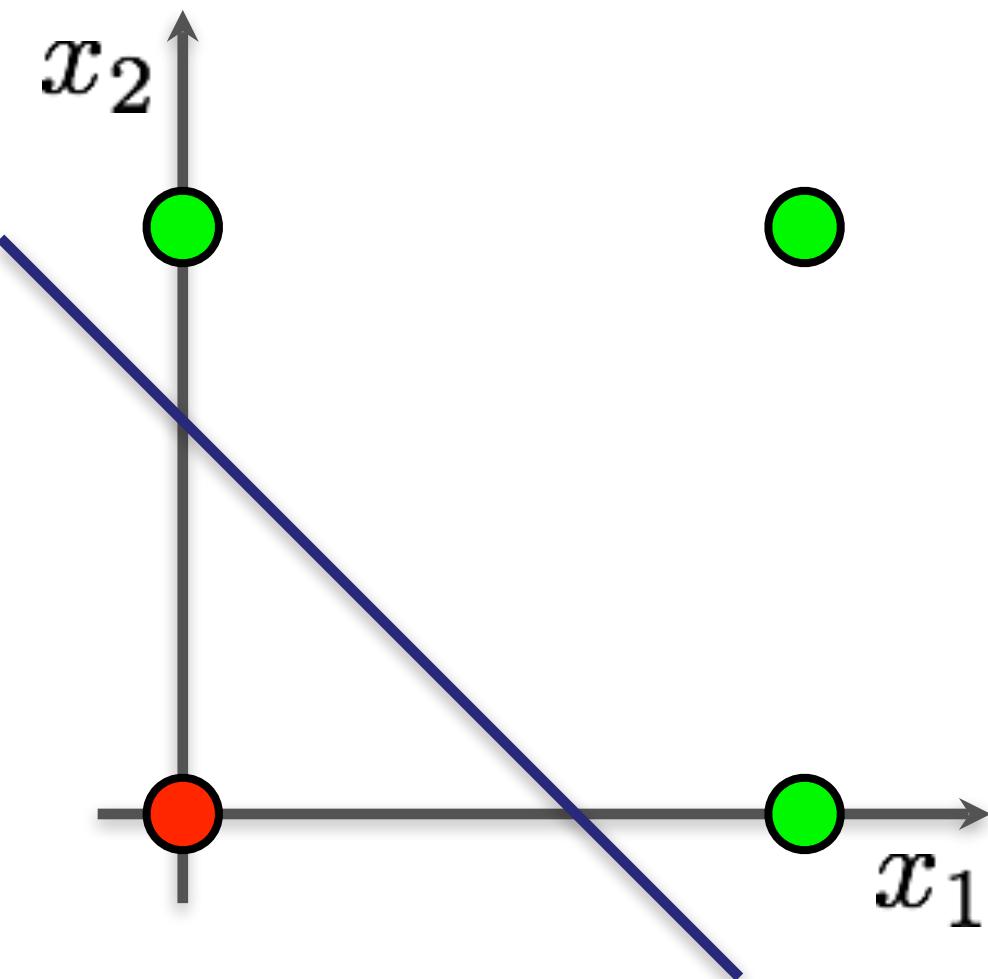
XOR Problem

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



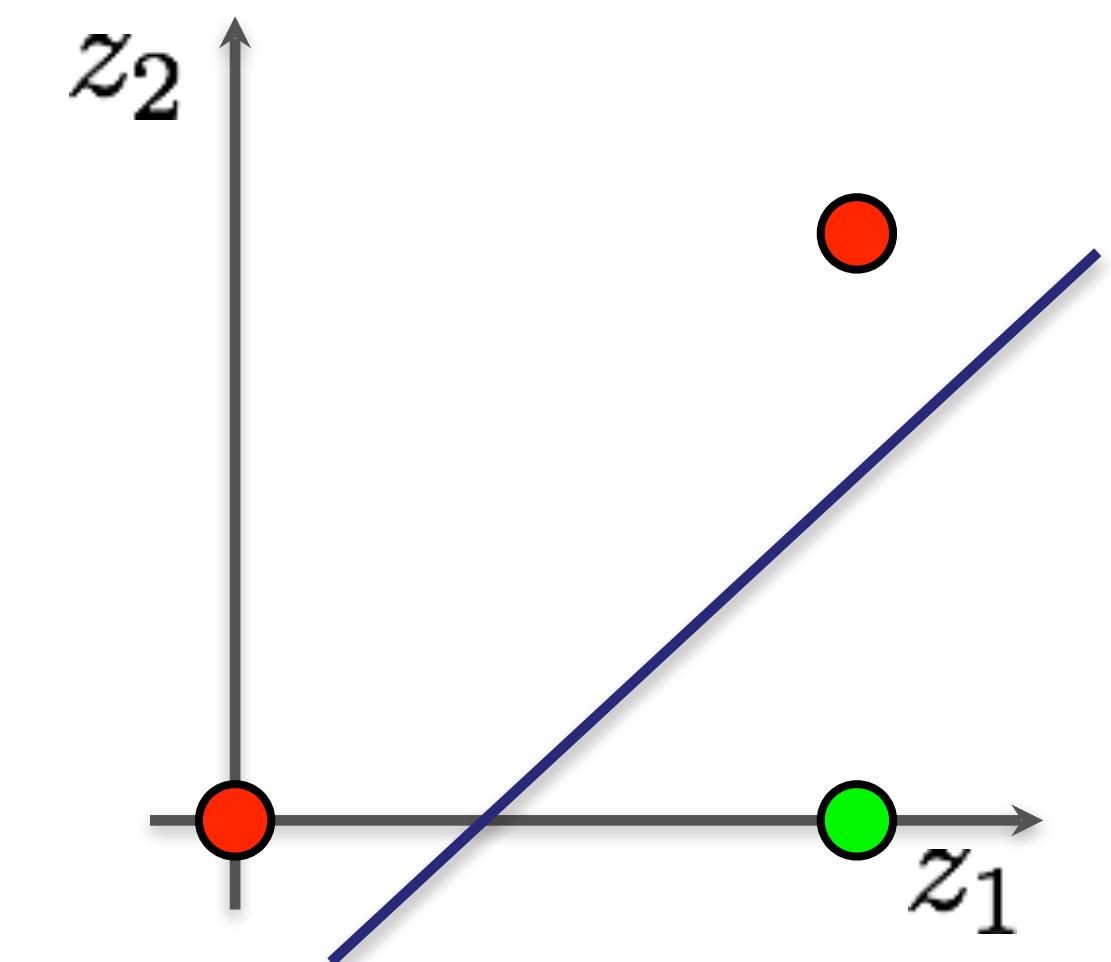
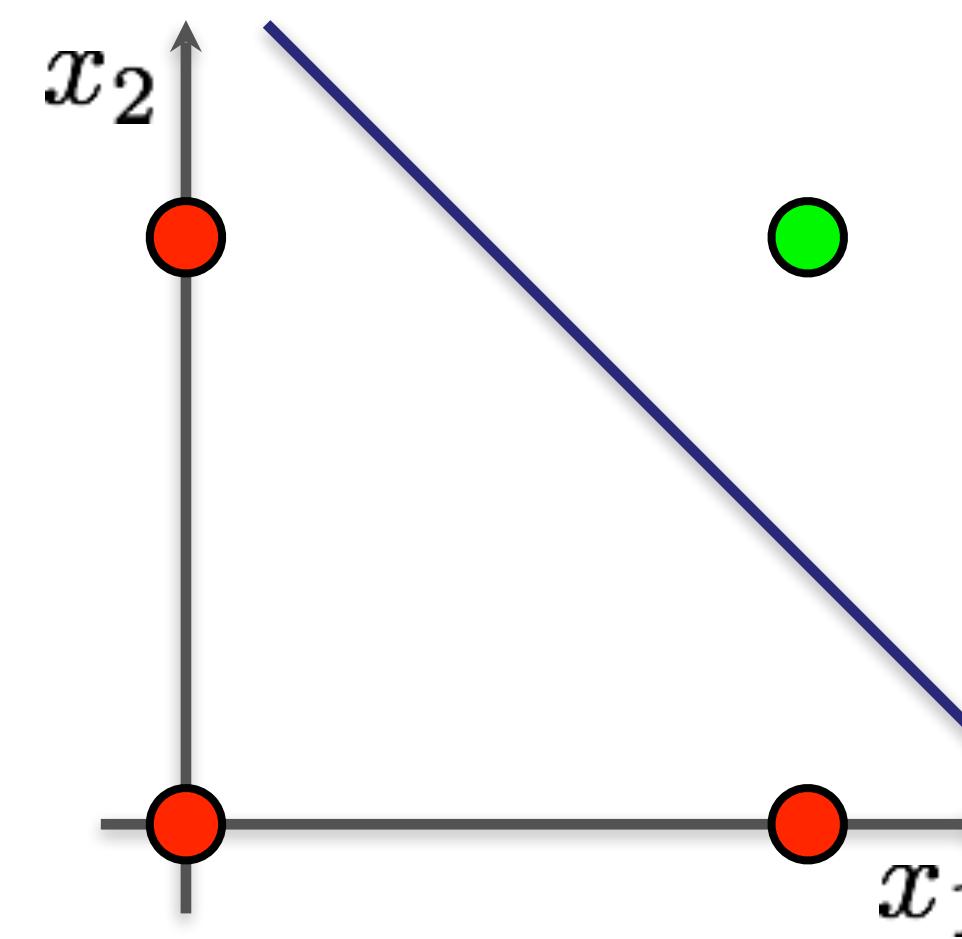
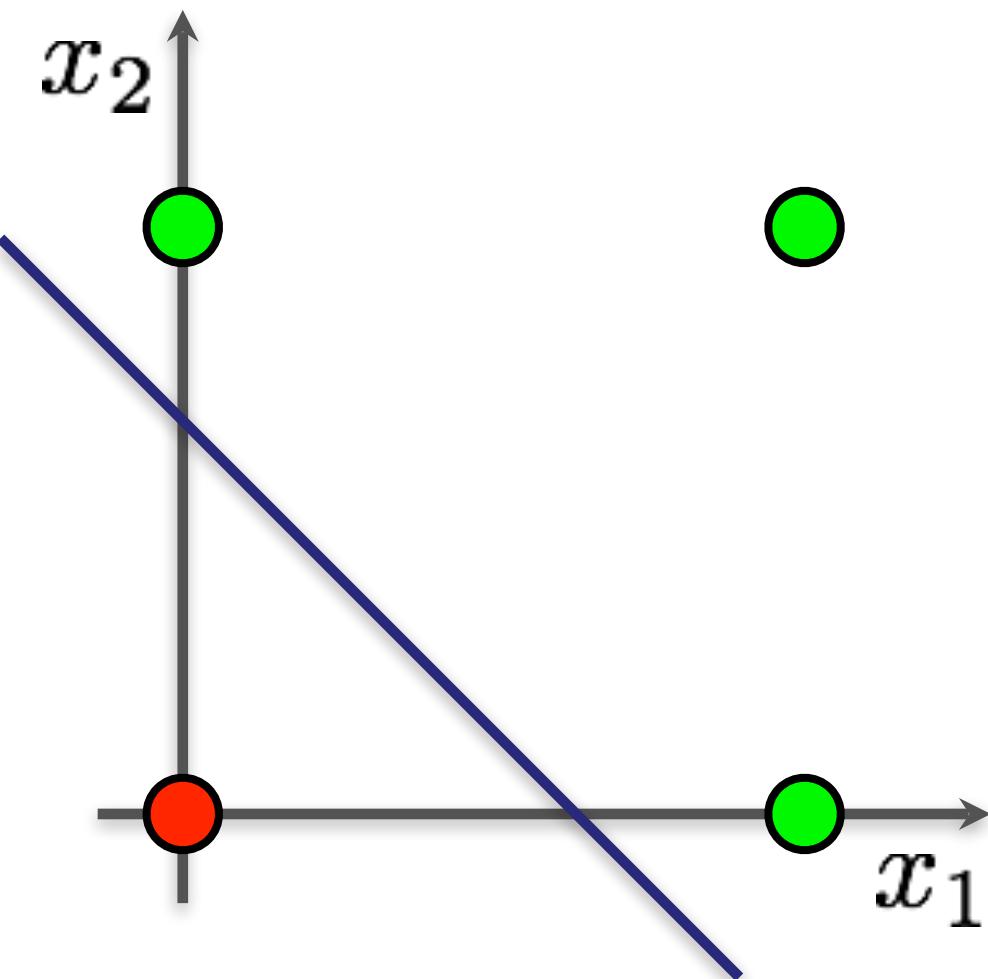
XOR Problem

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



XOR Problem

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



XOR Problem

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

XOR Problem

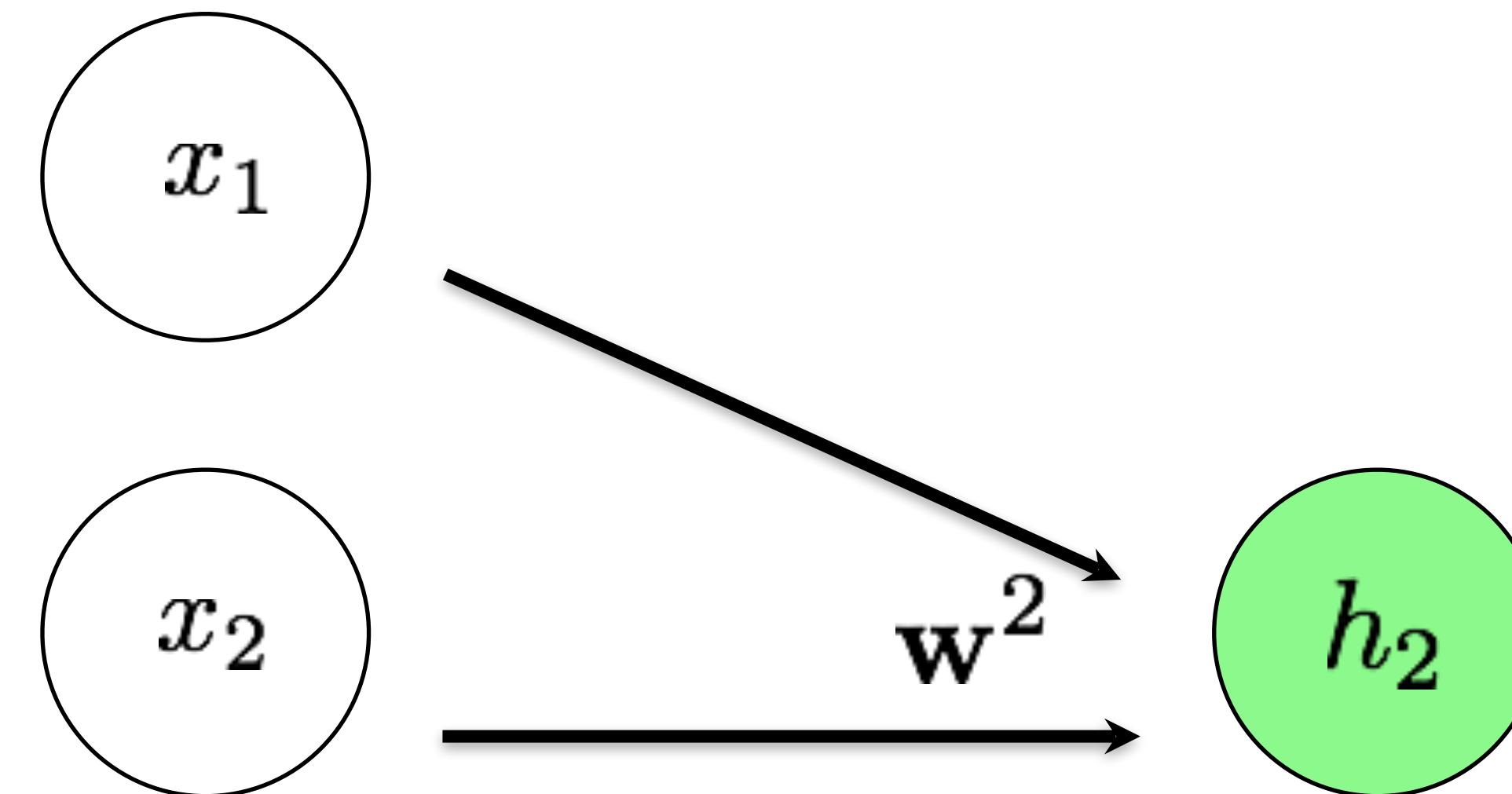
$$\begin{aligned}y &= f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \\&= f\left(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)\right) \\&= \mathcal{H}(\mathbf{w}^3, \boxed{\mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2))}, \boxed{\mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2))})\end{aligned}$$

x_1

x_2

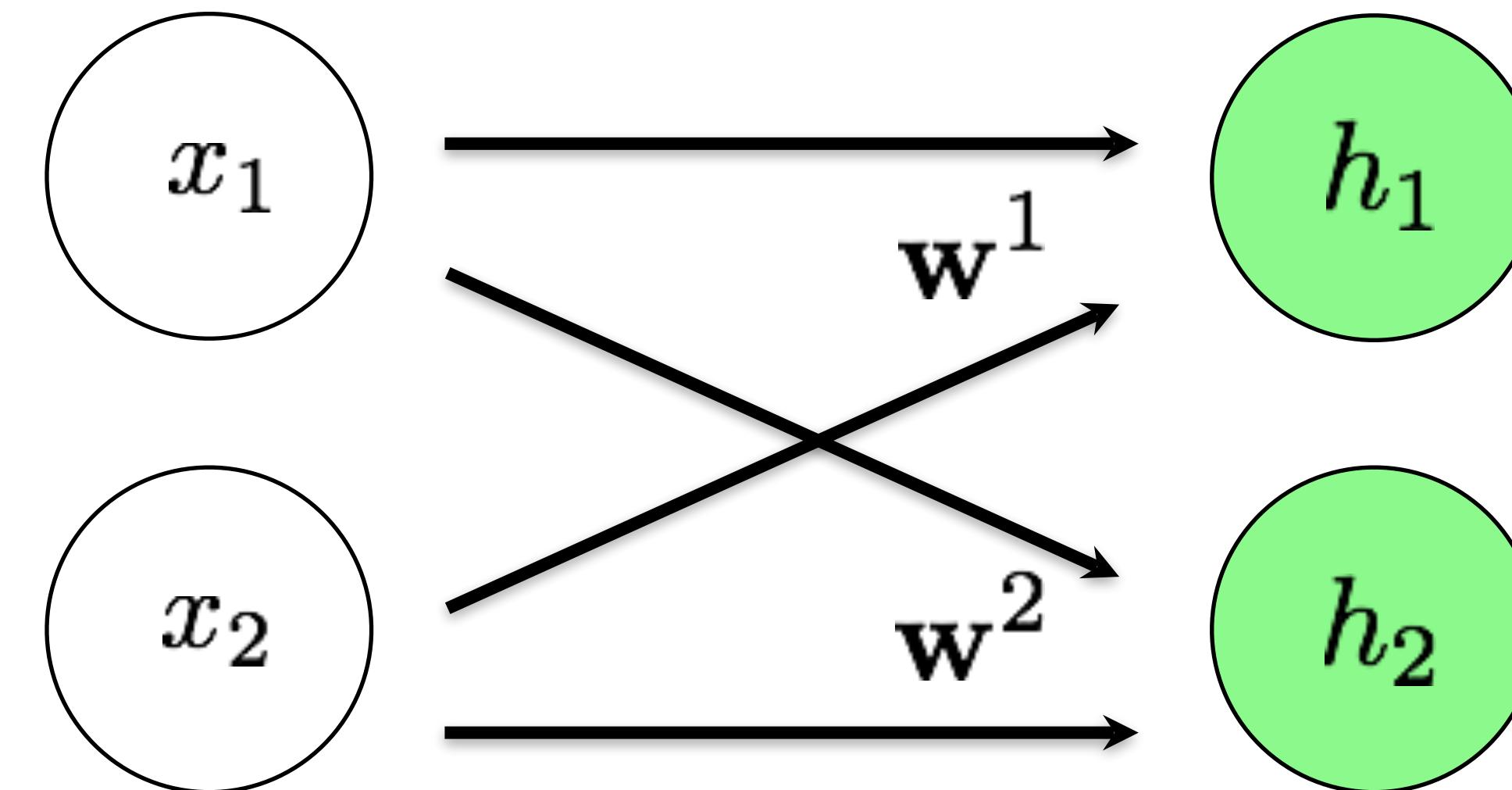
XOR Problem

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



XOR Problem

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



XOR Problem

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

