COMP0169: Machine Learning for Visual Computing

PCA and Linear Classifiers

Lectures will be Recorded

Principal Component Analysis

Principal Component Analysis

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

Principal Component Analysis

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

Principal Component Analysis

- **Problem setup** $\{\mathbf{x}_1, \ldots \mathbf{x}_n\}$ **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** $\{\mathbf{x}_1, \ldots \mathbf{x}_n\}$ **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** $\{\mathbf{x}_1, \ldots \mathbf{x}_n\}$ **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** $\{\mathbf{x}_1, \ldots \mathbf{x}_n\}$ **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

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

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

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

$$y = w_1 x_1 + w_2 x_2 + \ldots w_p x_p + b$$

Bias Trick

$$y = w_1 x_1 + w_2 x_2 + \ldots w_p x_p + b$$

$$= b.1 + w_1 x_1 + w_2 x_2 + \ldots w_p x_p$$

Bias Trick

$$y = w_1 x_1 + w_2 x_2 + \ldots w_p x_p + b$$

$$= b.1 + w_1 x_1 + w_2 x_2 + \ldots w_p x_p$$

$$=< [b\ w_1\ \ldots w_p], [1\ x_1\ \ldots x_p] >$$

Bias Trick

$$y = w_1 x_1 + w_2 x_2 + \ldots w_p x_p + b$$

$$= b.1 + w_1 x_1 + w_2 x_2 + \ldots w_p x_p$$

$$=< [b\ w_1\ \ldots w_p], [1\ x_1\ \ldots x_p] >$$

$$=< \mathbf{w}, \mathbf{x} >$$

Bias Trick

$$
\begin{aligned}
y &= w_1 x_1 + w_2 x_2 + \ldots w_p x_p + b \\
&= b.1 + w_1 x_1 + w_2 x_2 + \ldots w_p x_p \\
&= < [b\ w_1\ \ldots w_p], [1\ x_1\ \ldots x_p] > \\
&= < \mathbf{w}, \mathbf{x} > \\
&= \mathbf{w}^T \mathbf{x}
\end{aligned}
$$

Loss function and Optimization

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

Loss function and Optimization

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

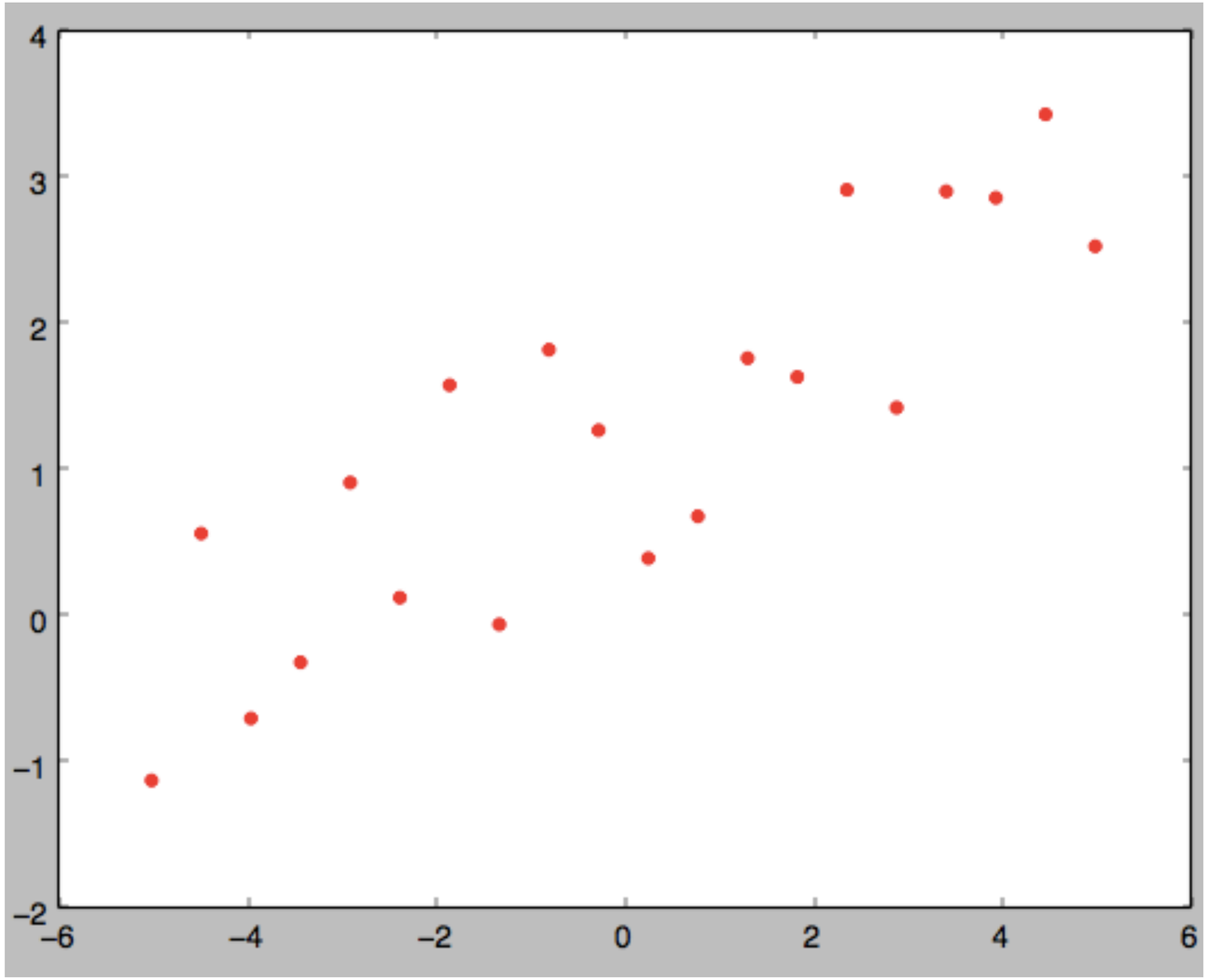
$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

Loss function and Optimization

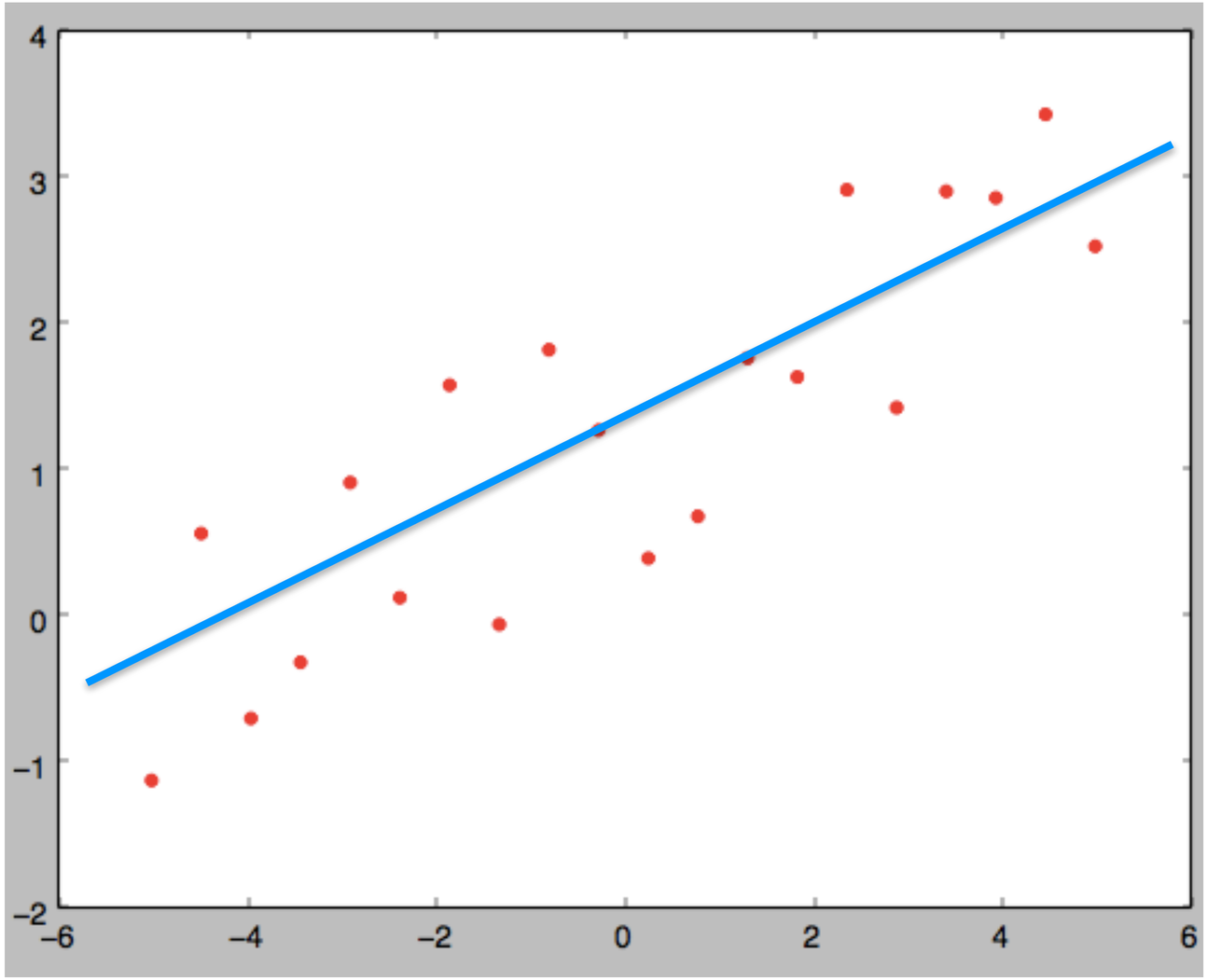
$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}^i - y^i)$$

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

Regression: Continuous Output



Regression: Continuous Output



Normal Equation (derivation)

Line Fitting

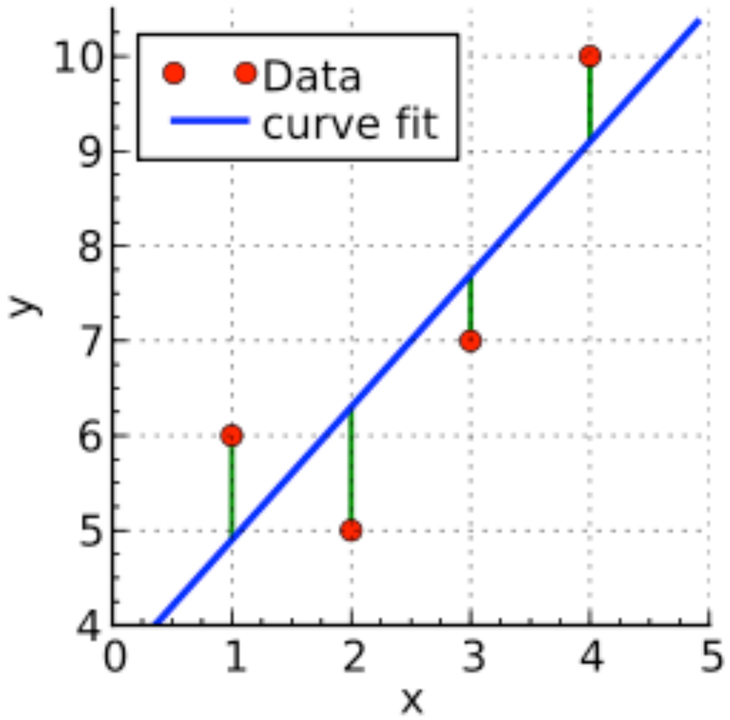
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Line Fitting

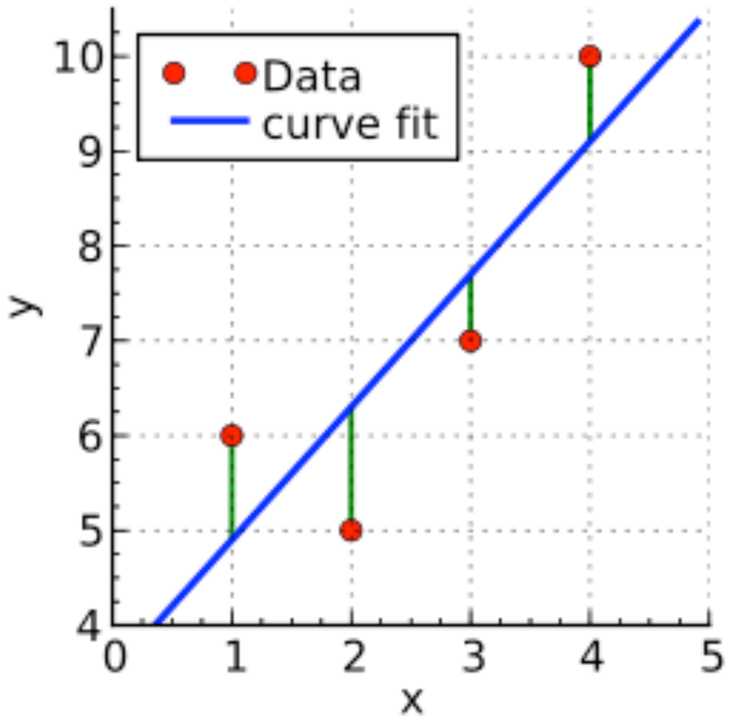
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

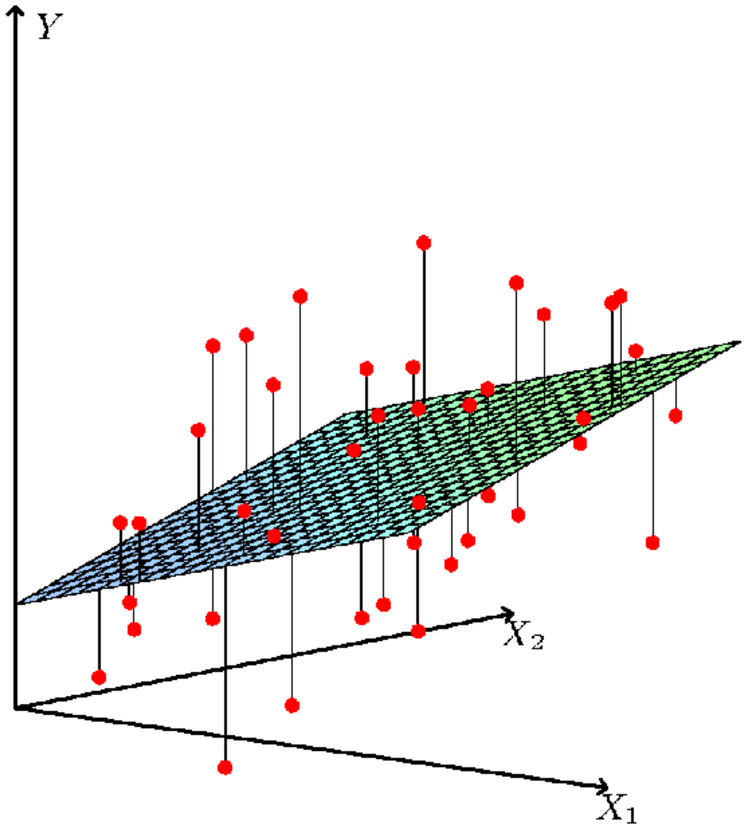
$$\mathbf{X} = \begin{bmatrix} x_0^1 & x_1^1 \\ \vdots & \vdots \\ x_0^N & x_1^N \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix}$$

Linear Regression (Line/Plane Fitting)

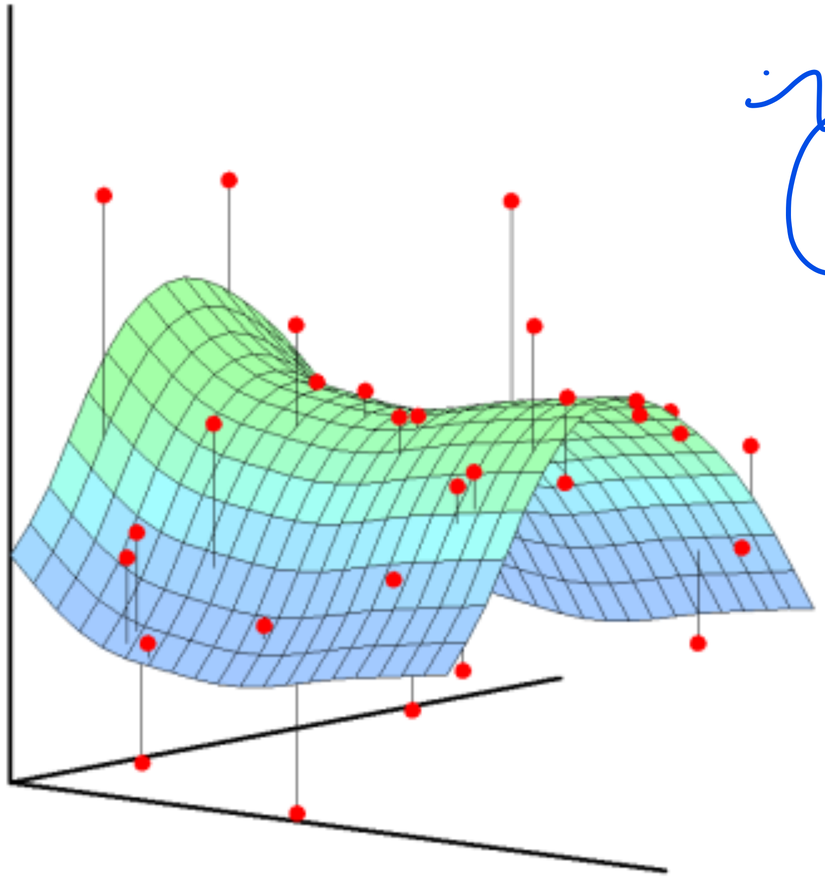


Linear Regression (Line/Plane Fitting)





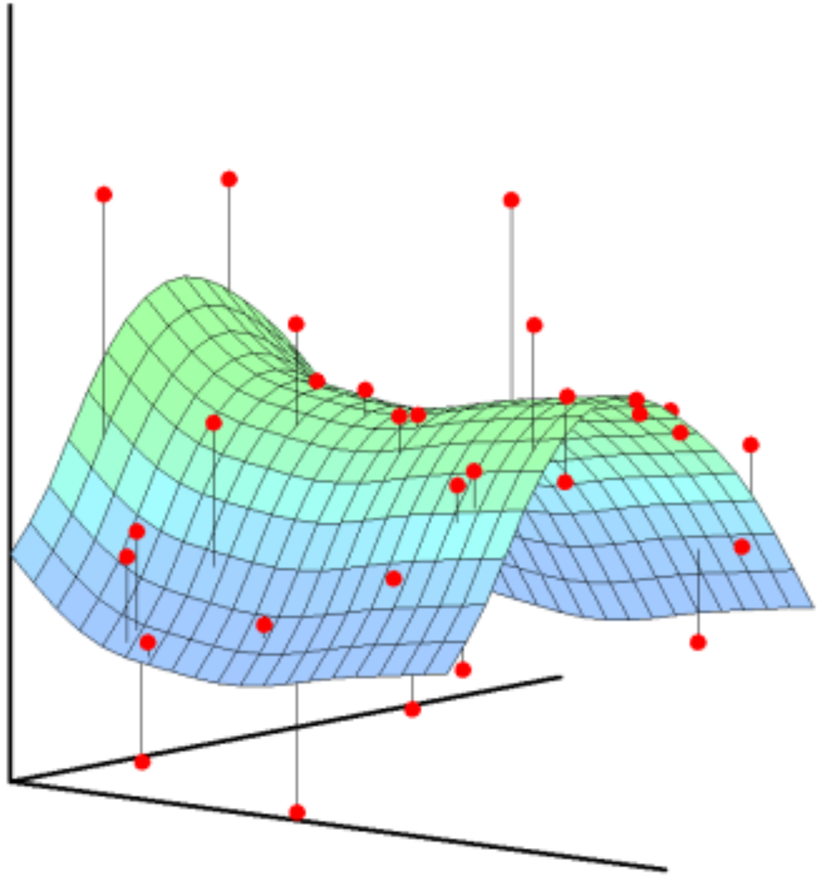
Polynomial Linear Regression



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

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

Polynomial Linear Regression

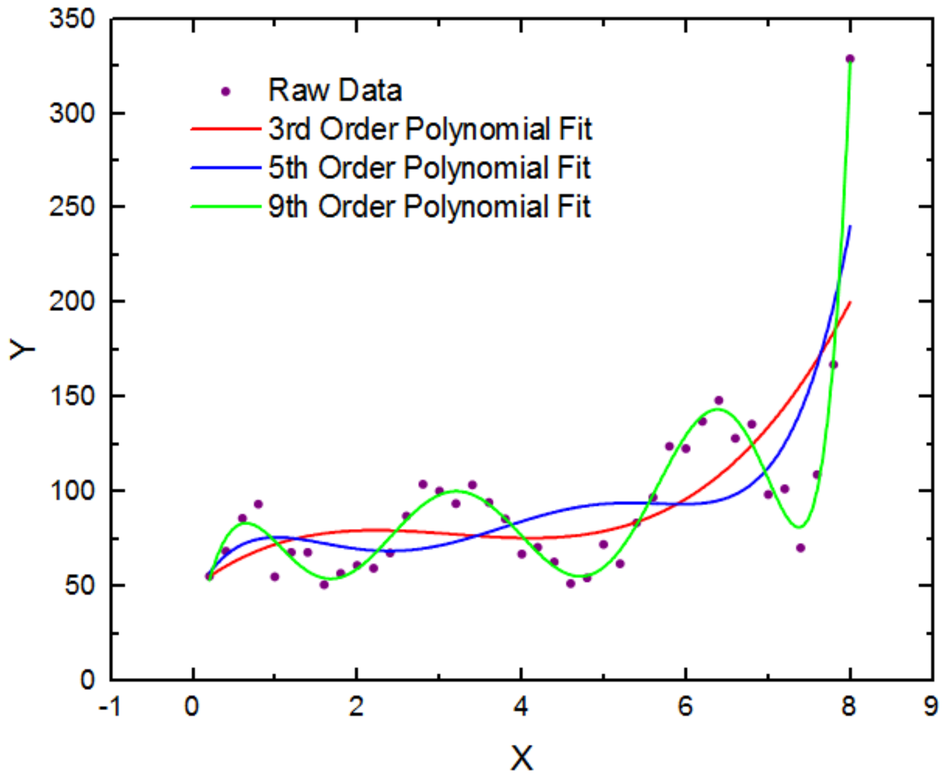


$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}$$

known nonlinearity

1D Example: k-th Degree Polynomial Fitting

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



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

Generalized Linear Regression

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

Generalized Linear Regression

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

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

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

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

LS Solution for Linear Regression

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

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

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

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

LS Solution for Generalized Linear Regression

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

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

LS Solution for Generalized Linear Regression

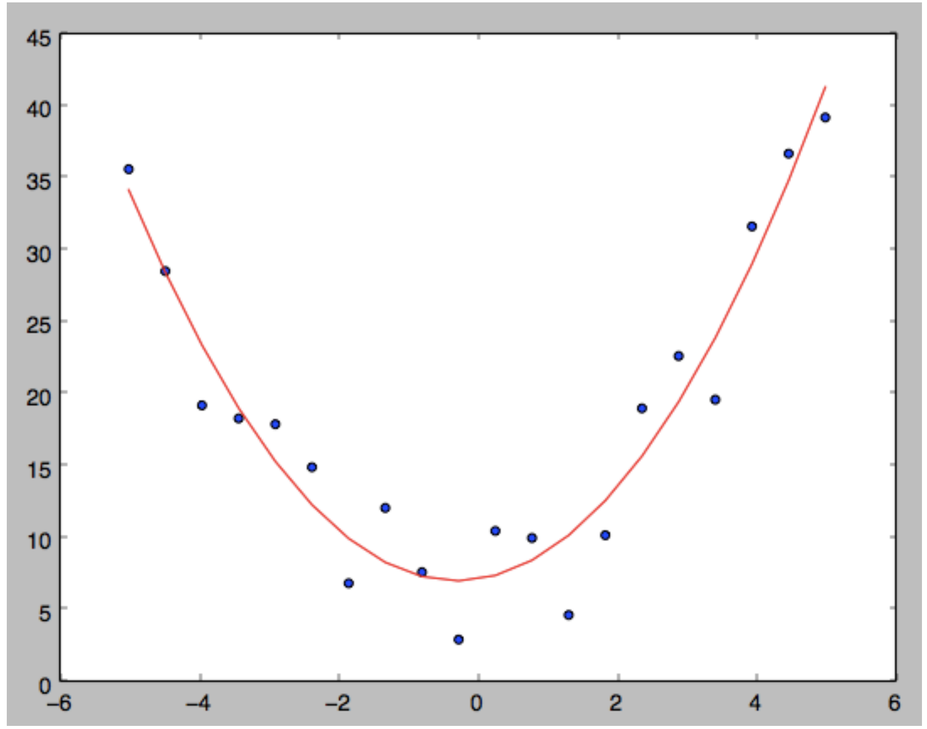
$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} + \boldsymbol{\epsilon}$$

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

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

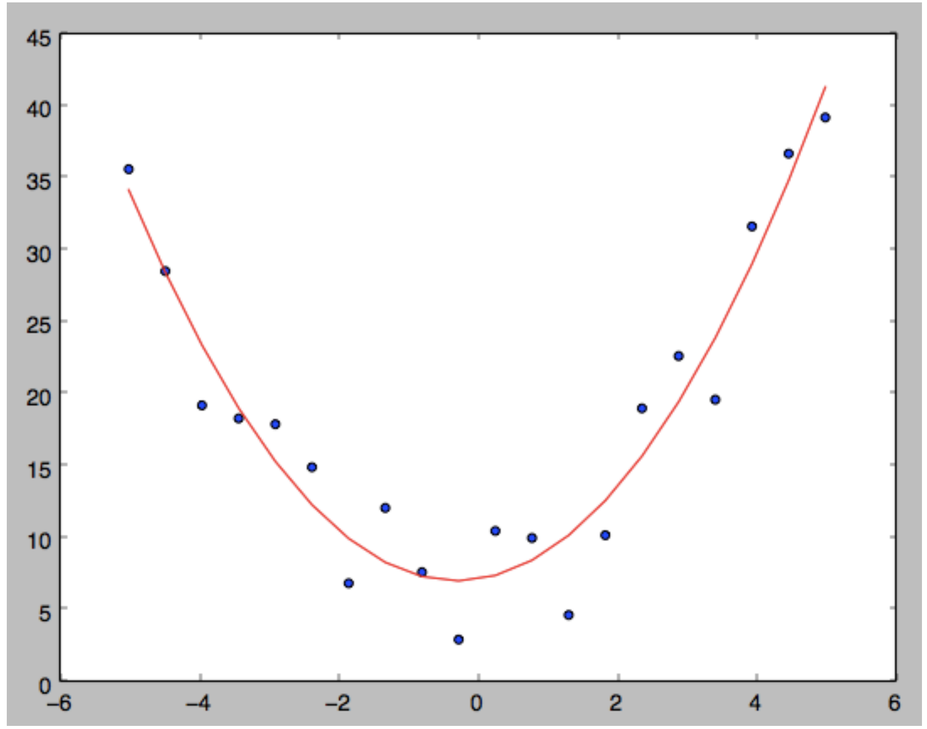
$$\mathbf{w}^* = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}\mathbf{y}$$

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

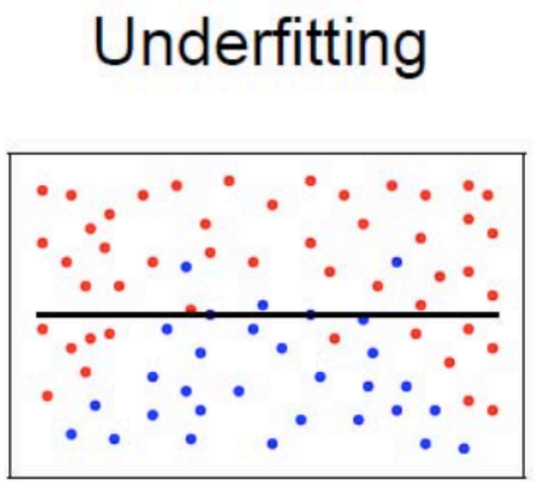



```
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# 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()
```

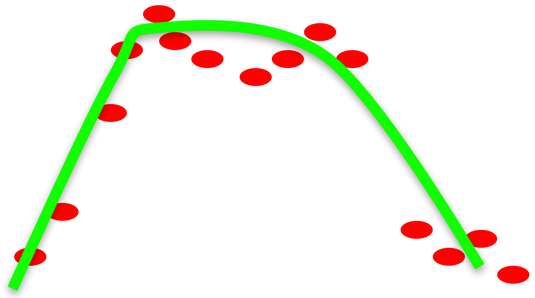
$$\mathbf{w}^* = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}\mathbf{y}$$

Underfitting vs. Overfitting

classification



regression

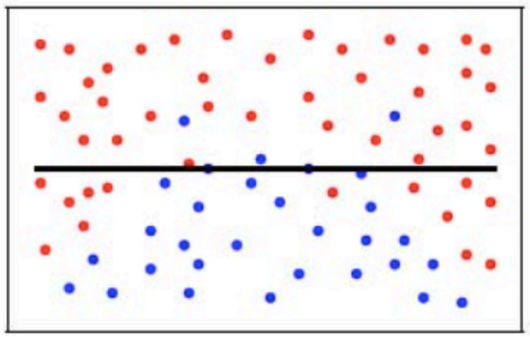


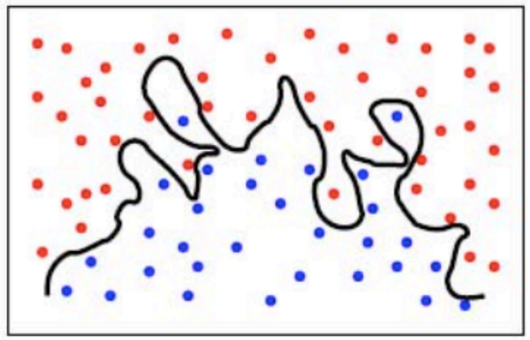
Underfitting vs. Overfitting

Underfitting

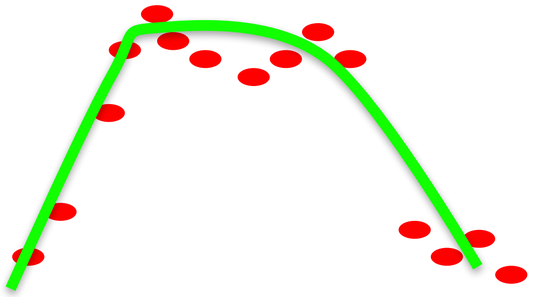
Overfitting

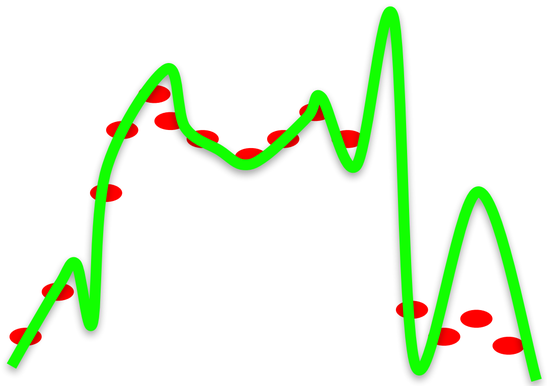
classification



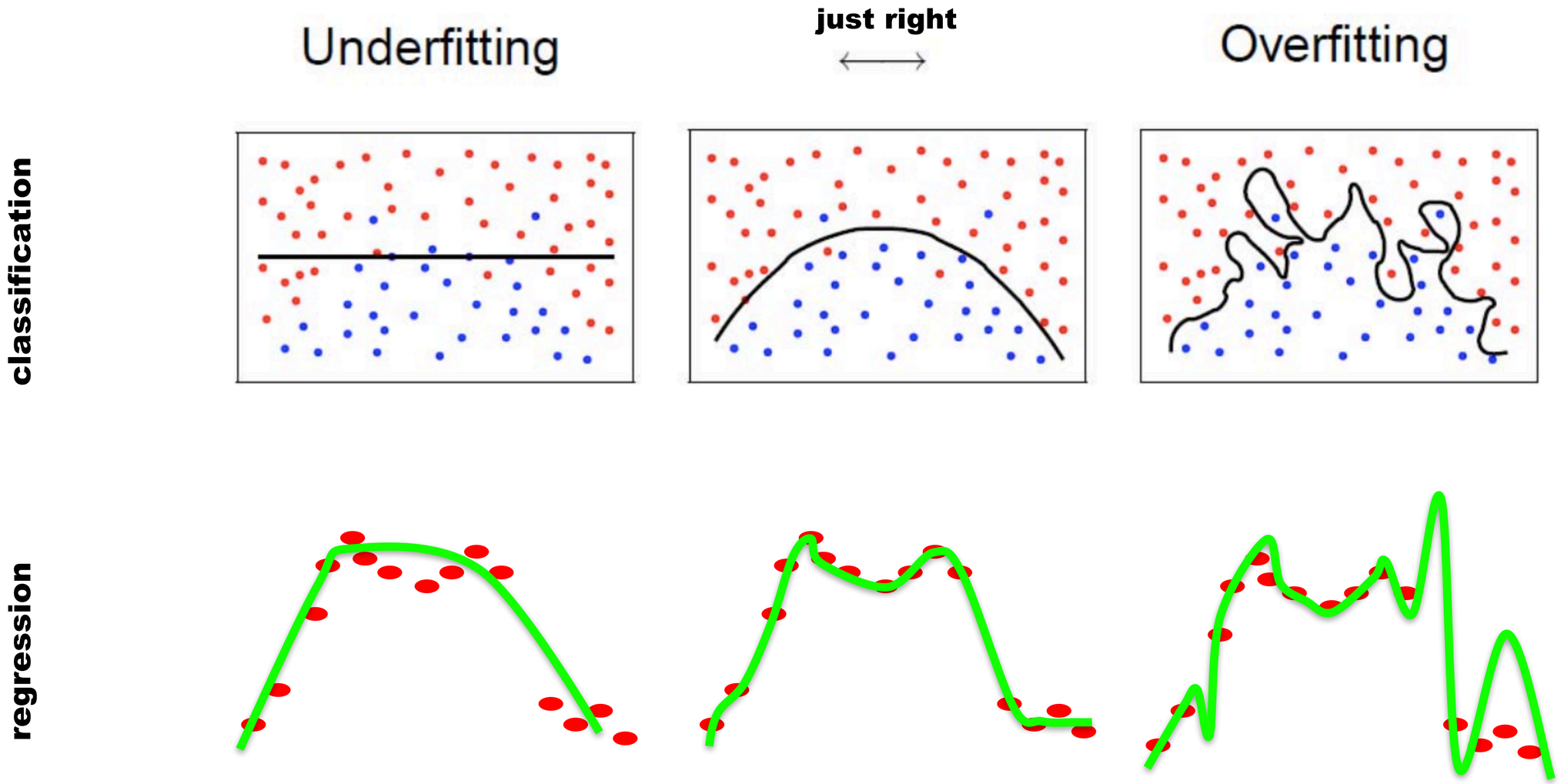


regression





Underfitting vs. Overfitting



Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

Regularized Linear Regression

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

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

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

Regularized Linear Regression

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

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

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

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

Regularized Linear Regression

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

residual vector

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

linear regression: minimize model error

Complexity term: (regularizer)

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

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

“data fidelity”

Regularized Linear Regression

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

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

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

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

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

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

"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: (regularizer)

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

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

"data fidelity" **complexity**

minimum remains to be determined

scalar, remains to be determined

Linear Binary Classifier

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

Linear Binary Classifier

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

Find weights $\mathbf{w}$ such that

Linear Binary Classifier

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

Find weights $\mathbf{w}$ such that

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

Linear Binary Classifier

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

Find weights $\mathbf{w}$ such that

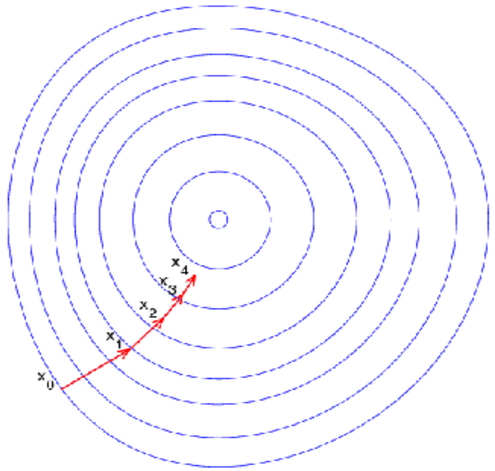
$$\begin{aligned} \mathbf{x}_i^T \mathbf{w} &\geq 0 \quad \text{if} \quad y_i = 1 \\ \mathbf{x}_i^T \mathbf{w} &< 0 \quad \text{if} \quad y_i = -1 \end{aligned}$$

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

Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

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

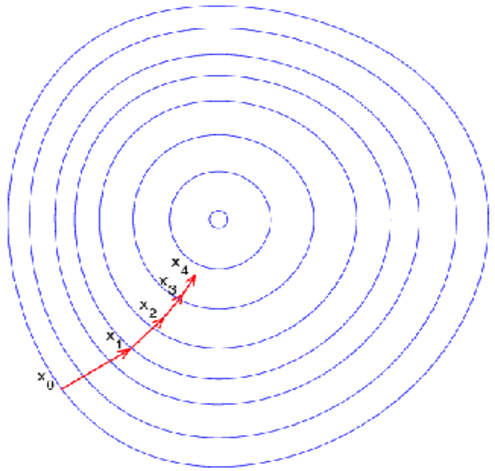


Gradient Descent Minimization

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.

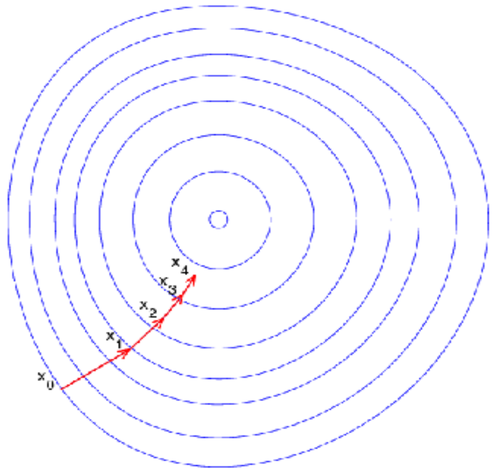


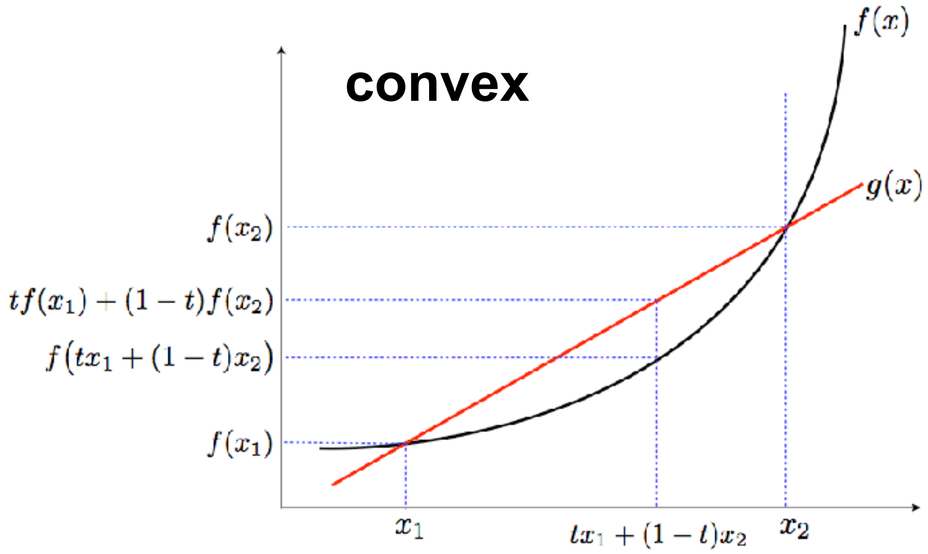
Gradient Descent Minimization

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.



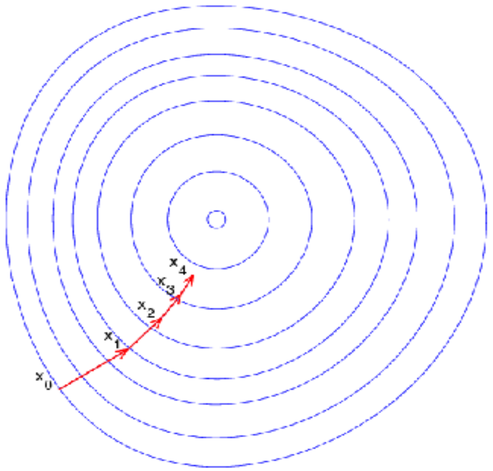


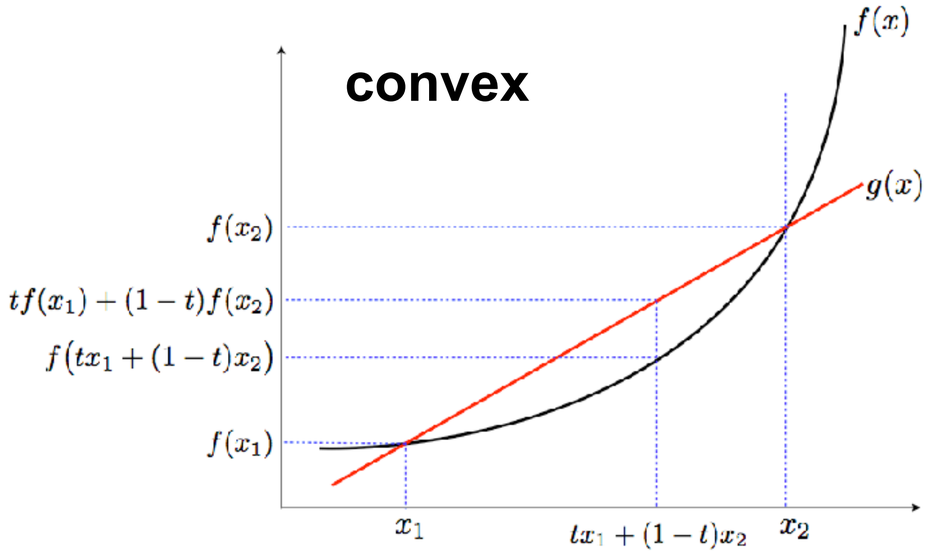
Gradient Descent Minimization

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.





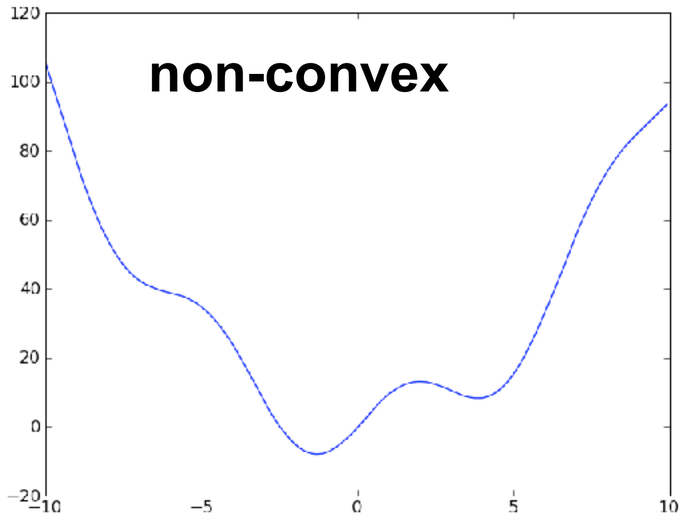


Image Classifier

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

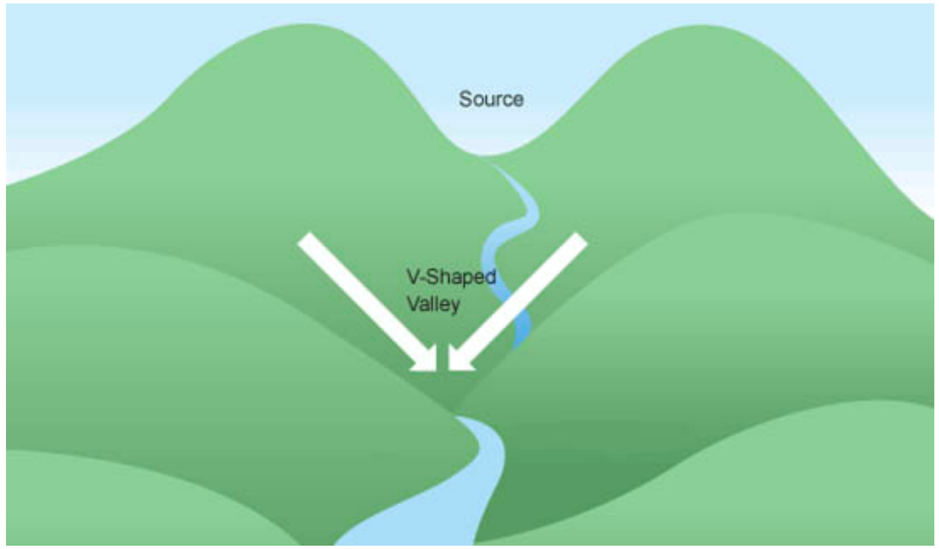
- **Data (training/validation/test)** $\{\mathbf{x}_i, y_i\}$
- **Scoring function (probability of belonging to class k)** $f(\mathbf{x}_i, \mathbf{W})$
- **Loss function** $L_i(W) := h(f(\mathbf{x}_i, W), y_i) \quad L(W) := \frac{1}{N} \sum_i L_i(W)$

Image Classifier

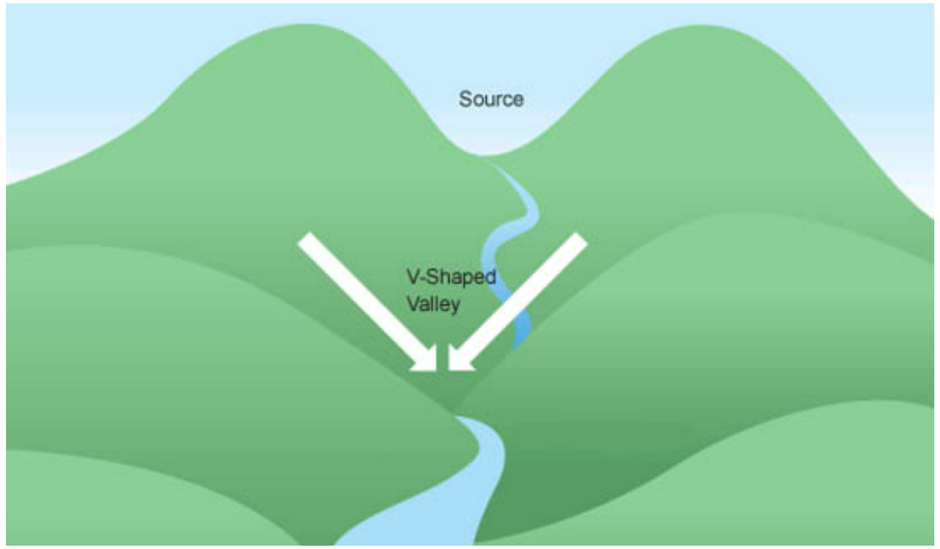
- **Data (training/validation/test)** $\{\mathbf{x}_i, y_i\}$
- **Scoring function (probability of belonging to class k)** $f(\mathbf{x}_i, \mathbf{W})$
- **Loss function** $L_i(W) := h(f(\mathbf{x}_i, W), y_i) \quad L(W) := \frac{1}{N} \sum_i L_i(W)$
- **Optimization** $\mathbf{W}^* := \arg\min L(W) = \arg\min L_i(W)/N$

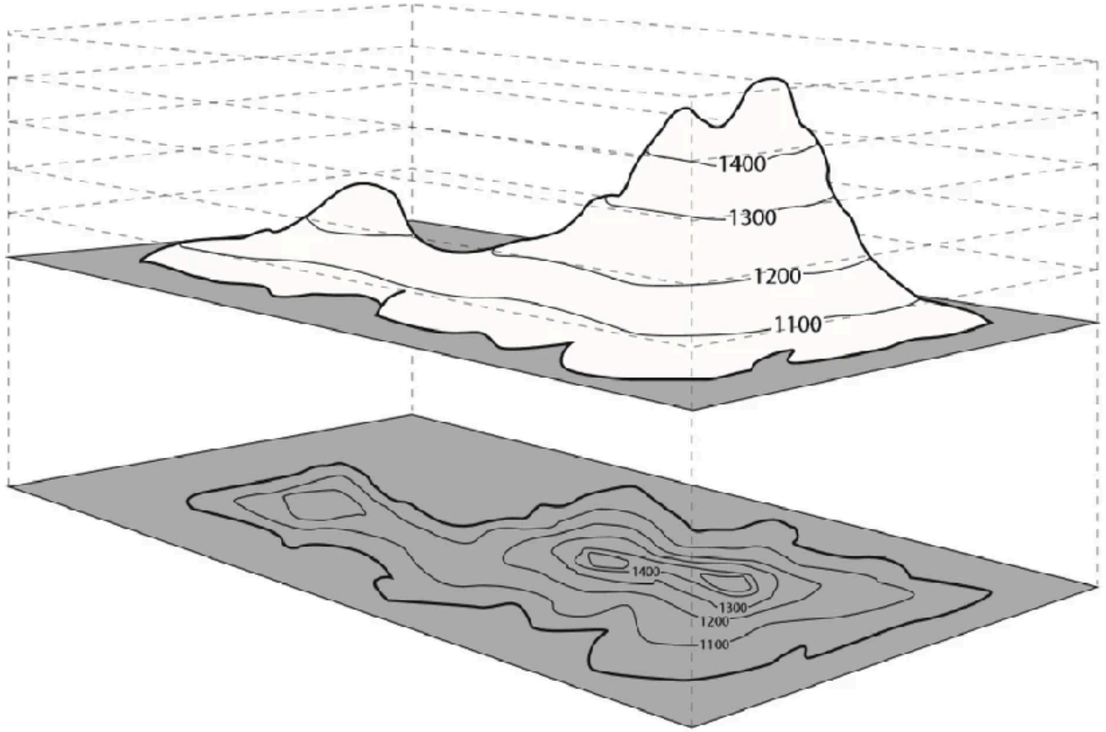
Error Landscape

Error Landscape

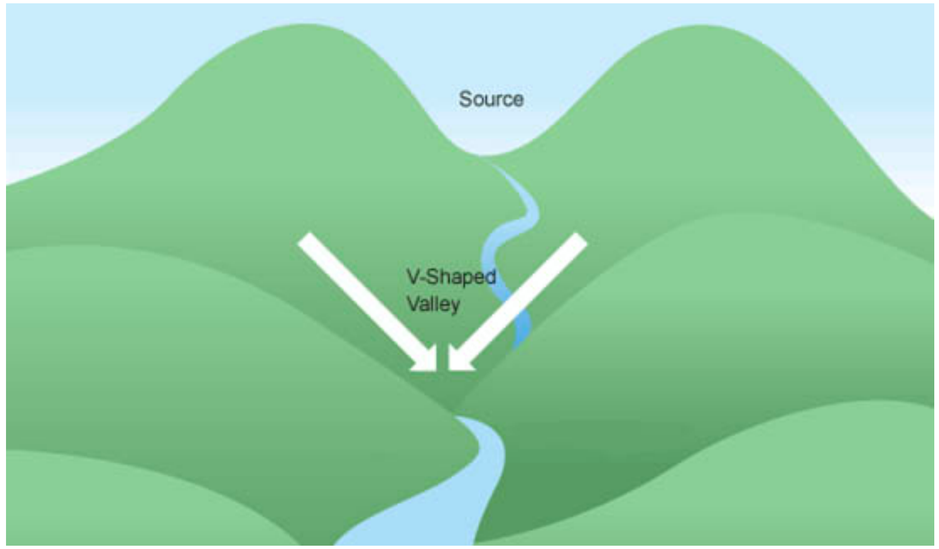


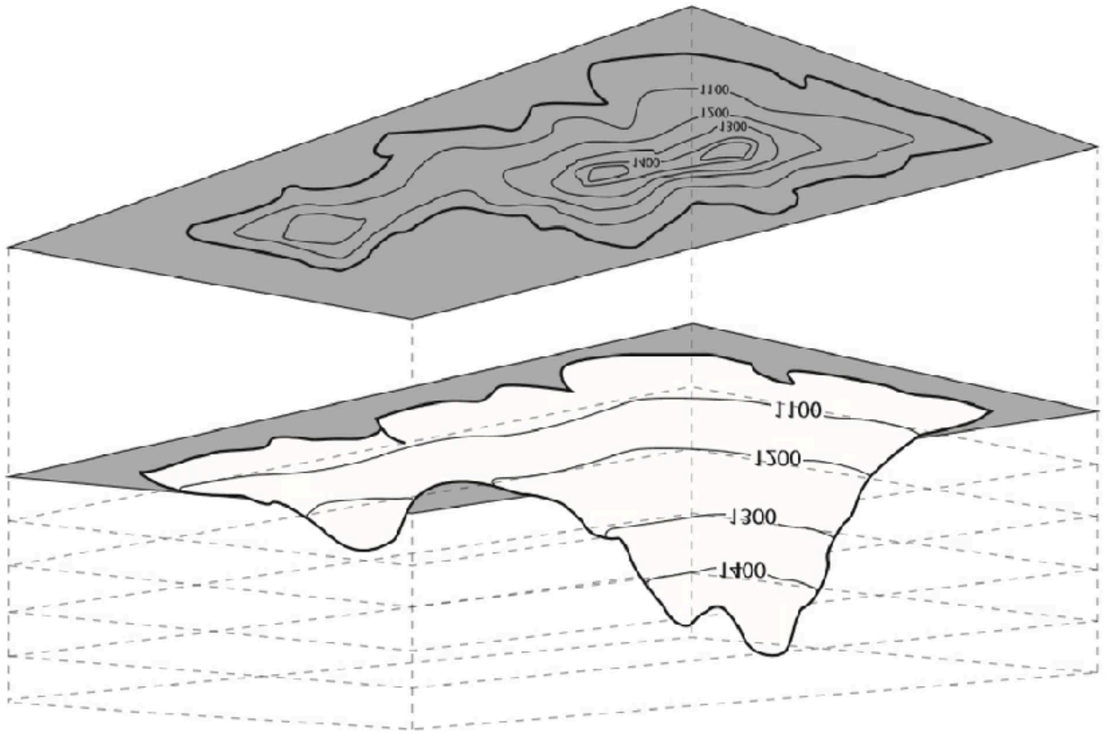
Error Landscape





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

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

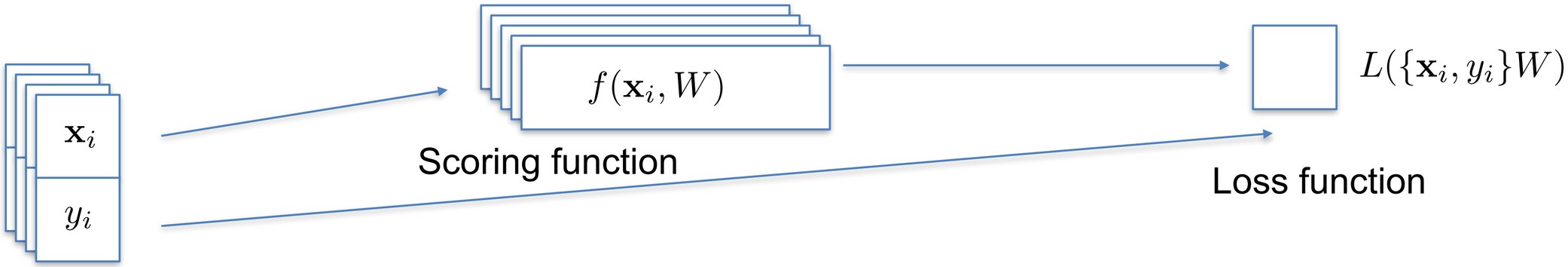
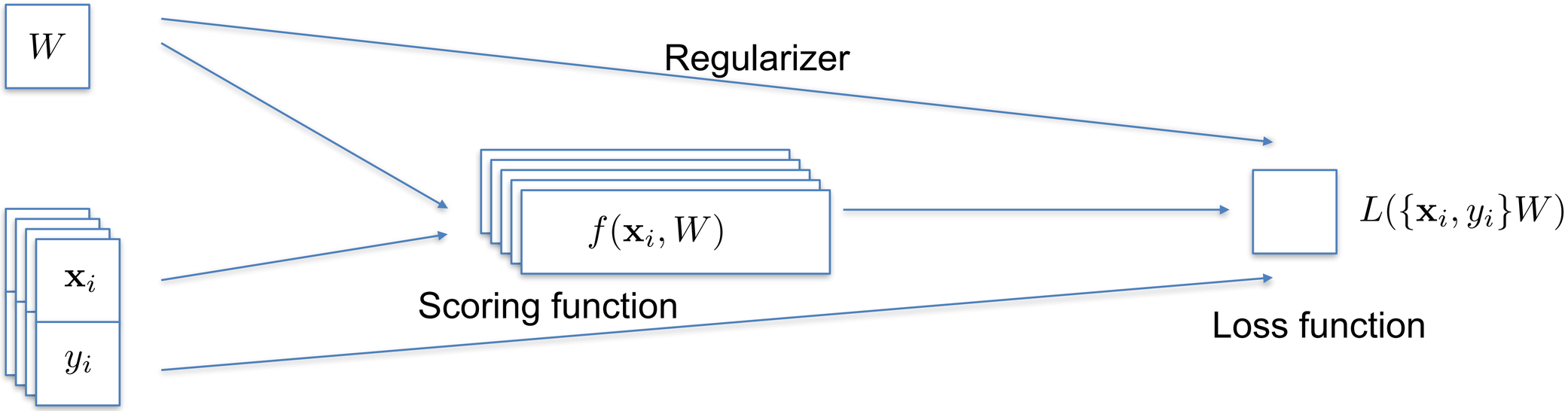


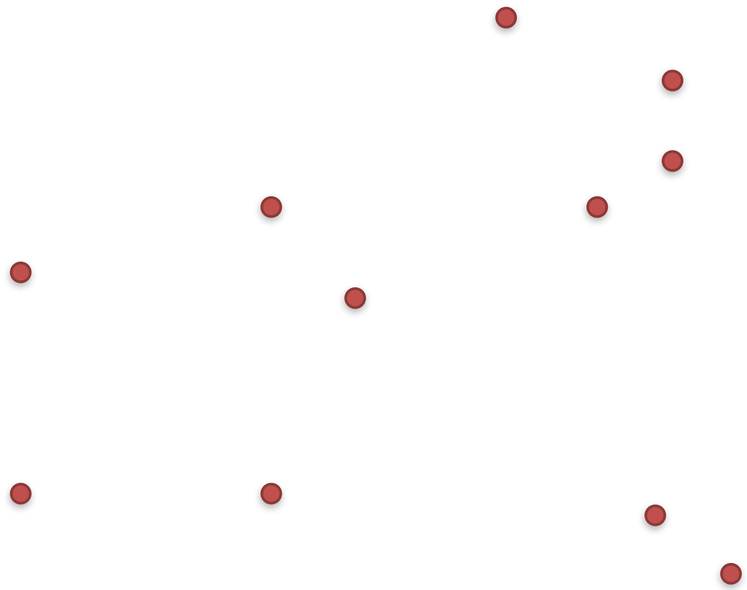
Image Classifier



Method 1a: Random search

- bestLoss = ∞
- Loop

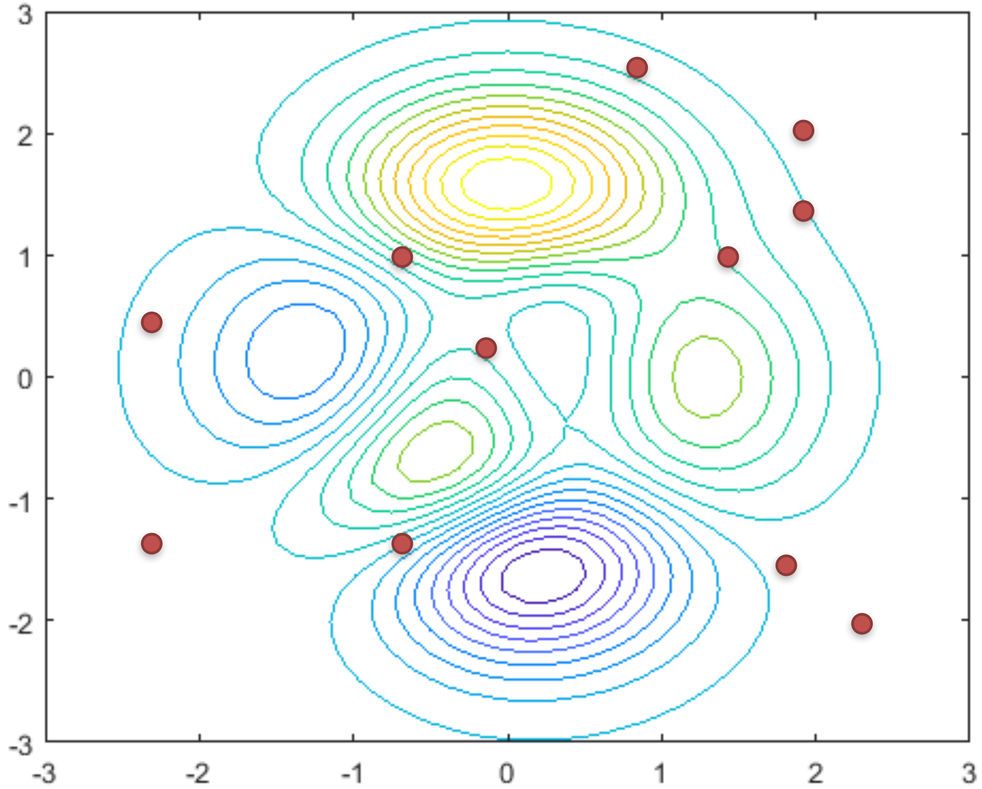
```
      Pick random W
      Compute loss
      if loss<bestLoss
            bestLoss = loss
            currW = W
       endif

endLoop
```



Method 1a: Random search

- `bestLoss = ∞`
- Loop

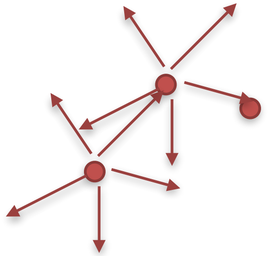
```
        Pick random W
        Compute loss
        if loss<bestLoss
              bestLoss = loss
              currW = W
         endif

  endLoop
```



Method 1b: Random search++

- bestLoss = ∞
- W = setRand
- stepsize
- Loop

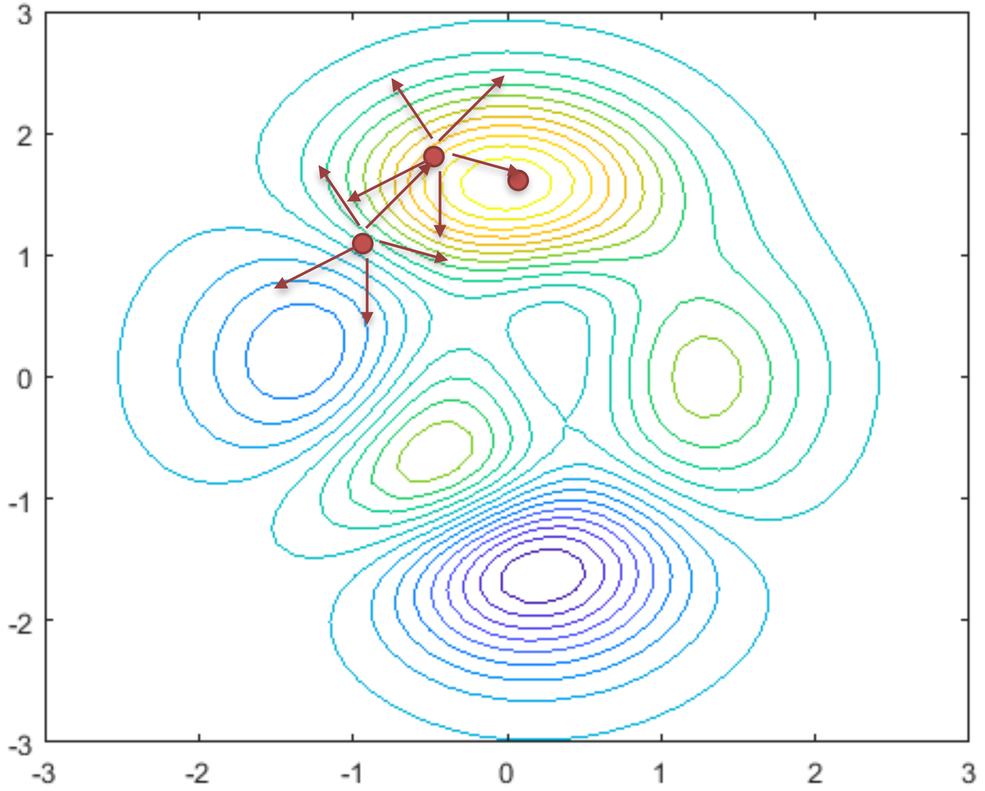
```
        W_local = W + stepsize*W_random
        Compute loss
        if loss<bestLoss
              bestLoss = loss
              W = W_local
         endif

  endLoop
```



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
```



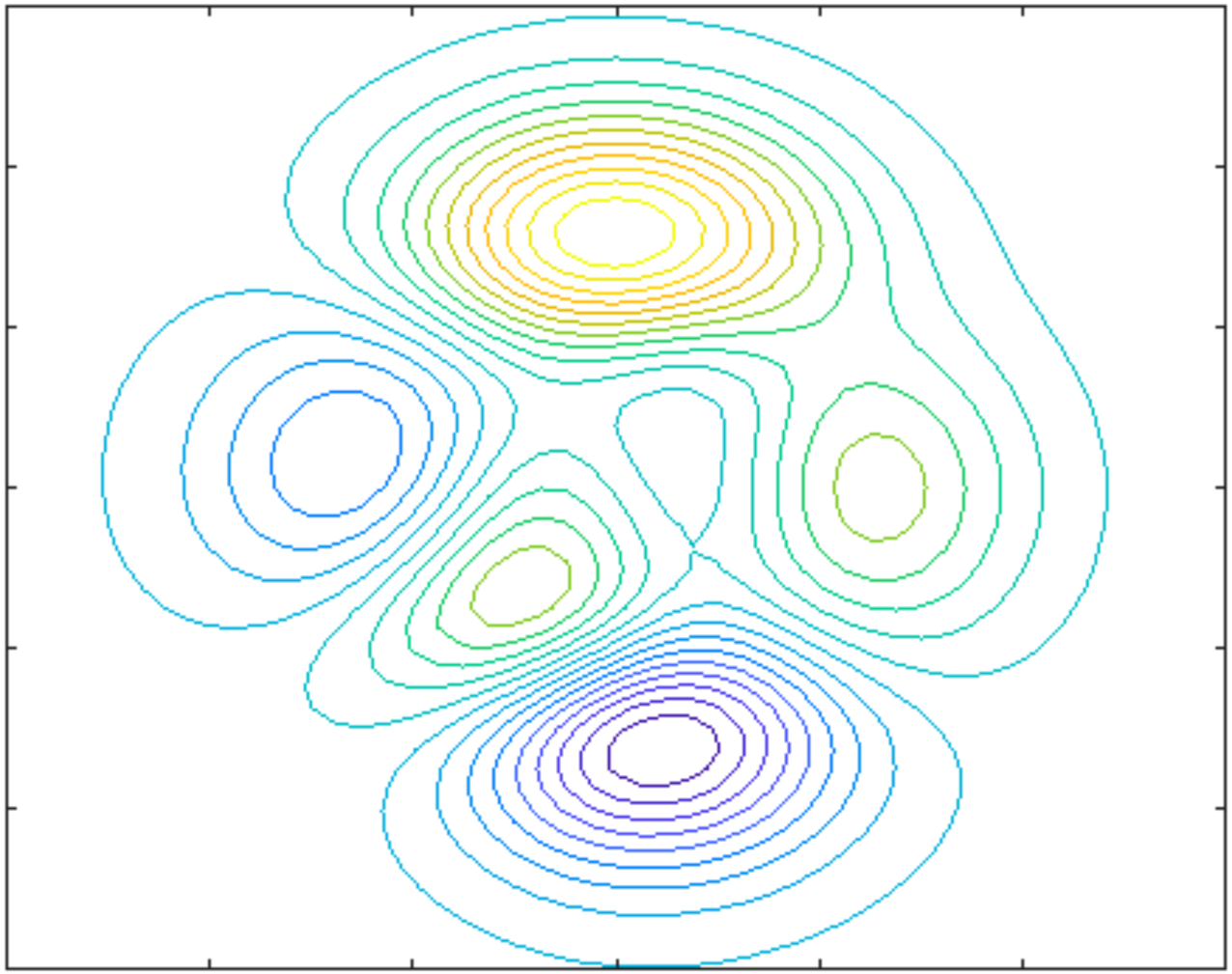
Optimize

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

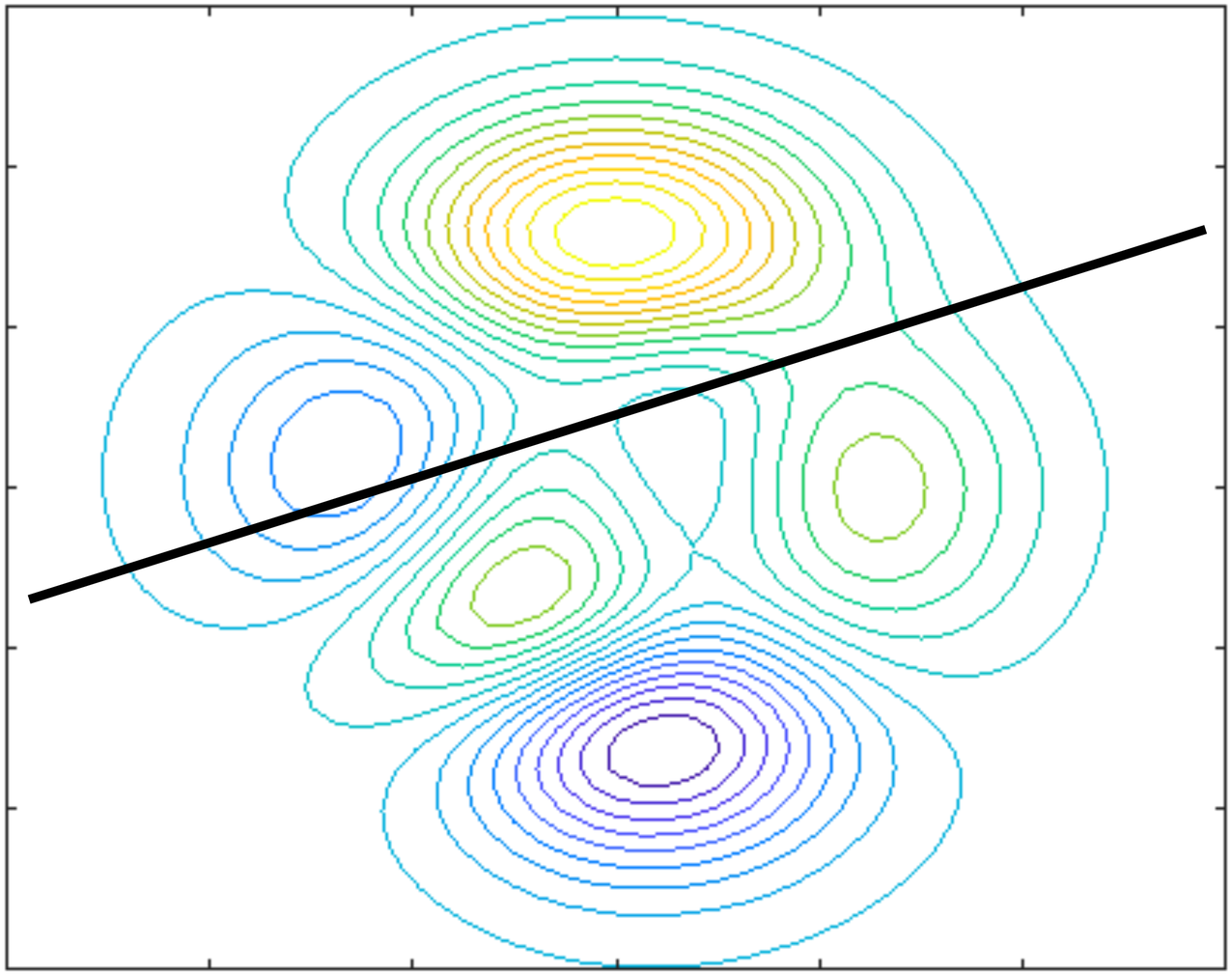
$$\nabla g(\mathbf{x}_i, W) = \left[\frac{\partial g(\mathbf{x}_i, W)}{w_i} = 0 \right]_{D \times 1}$$

Method 2: Follow Negative of Gradient

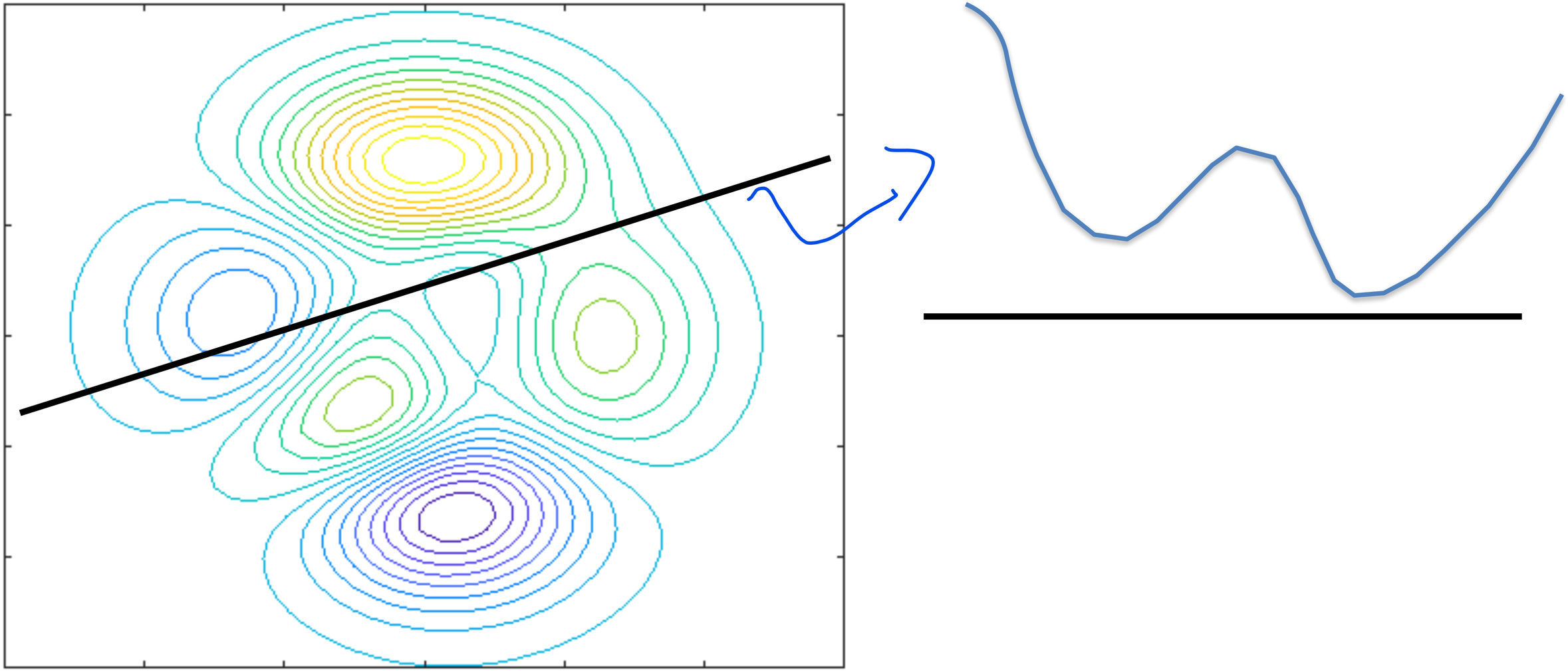
Method 2: Follow Negative of Gradient



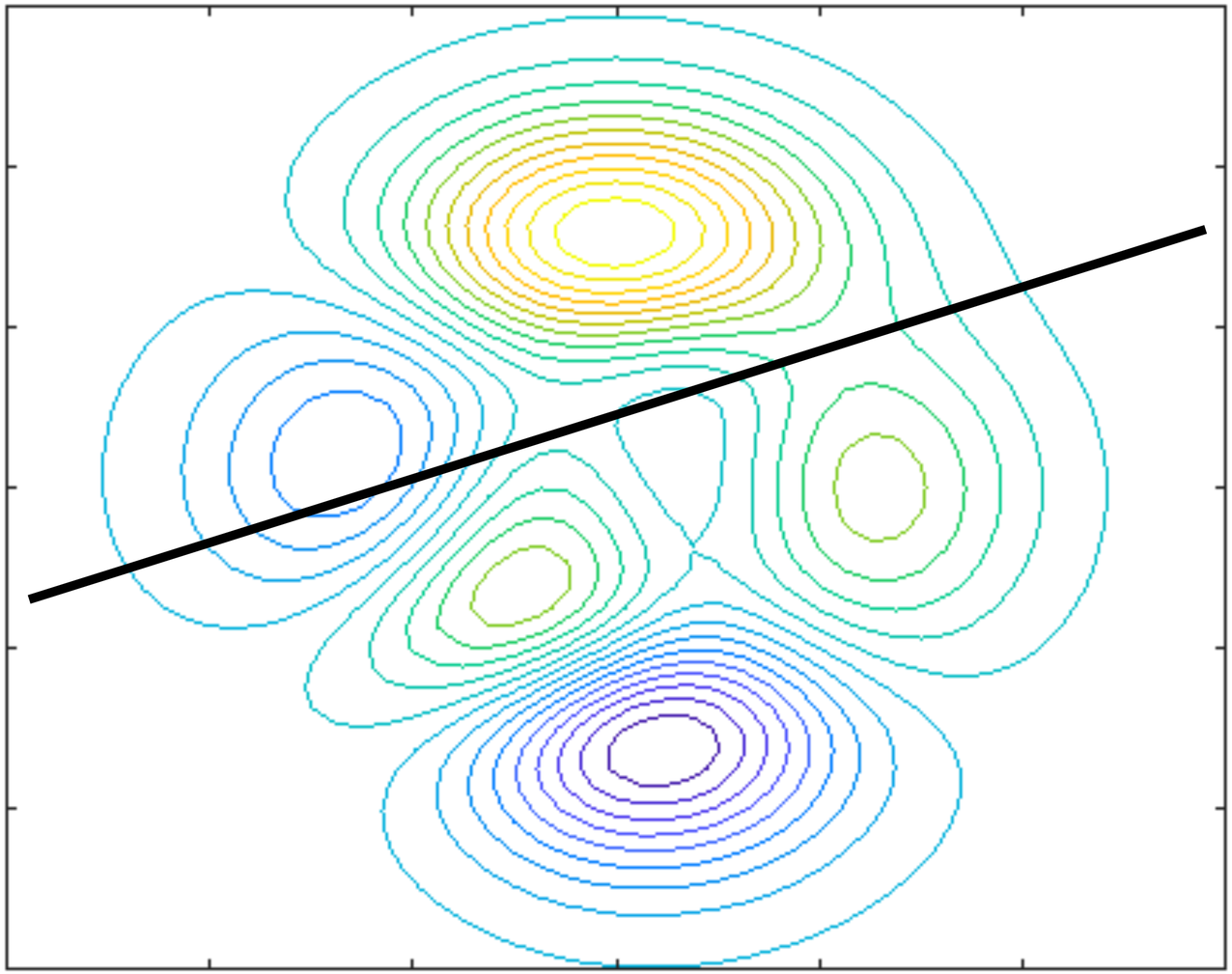
Method 2: Follow Negative of Gradient

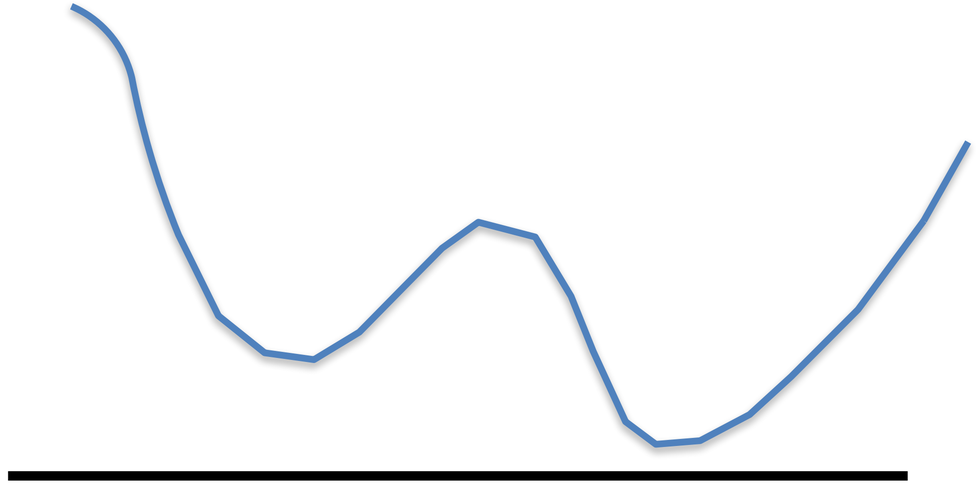


Method 2: Follow Negative of Gradient



Method 2: Follow Negative of Gradient

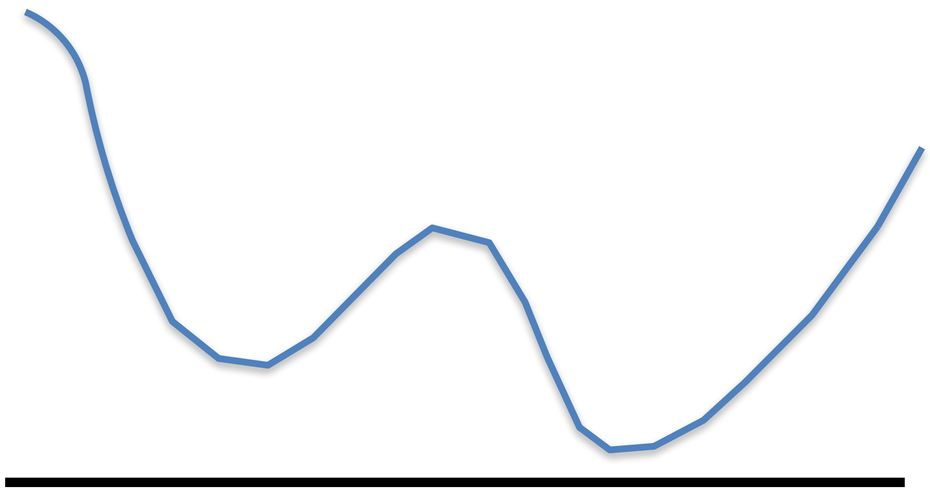




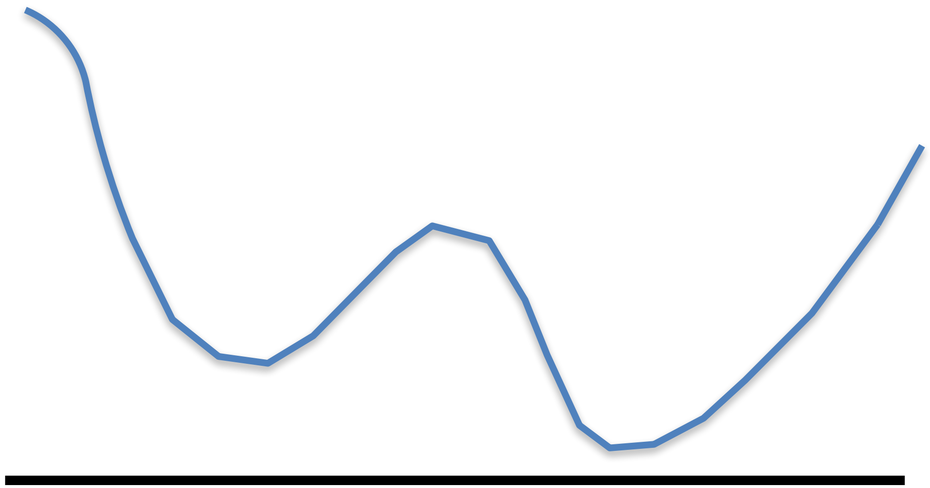
$$\frac{df(x)}{dx}$$

Method 2: Follow Negative of Gradient

Method 2: Follow Negative of Gradient



Method 2: Follow Negative of Gradient



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

Compute f(x), numerically

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

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

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

$$= \frac{1}{n}\sum_{i\in V} -y_i(\mathbf{x}_i^T\mathbf{w})$$

where V is the set of indices with $y_i(\mathbf{x}_i^T\mathbf{w}) < 0$

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

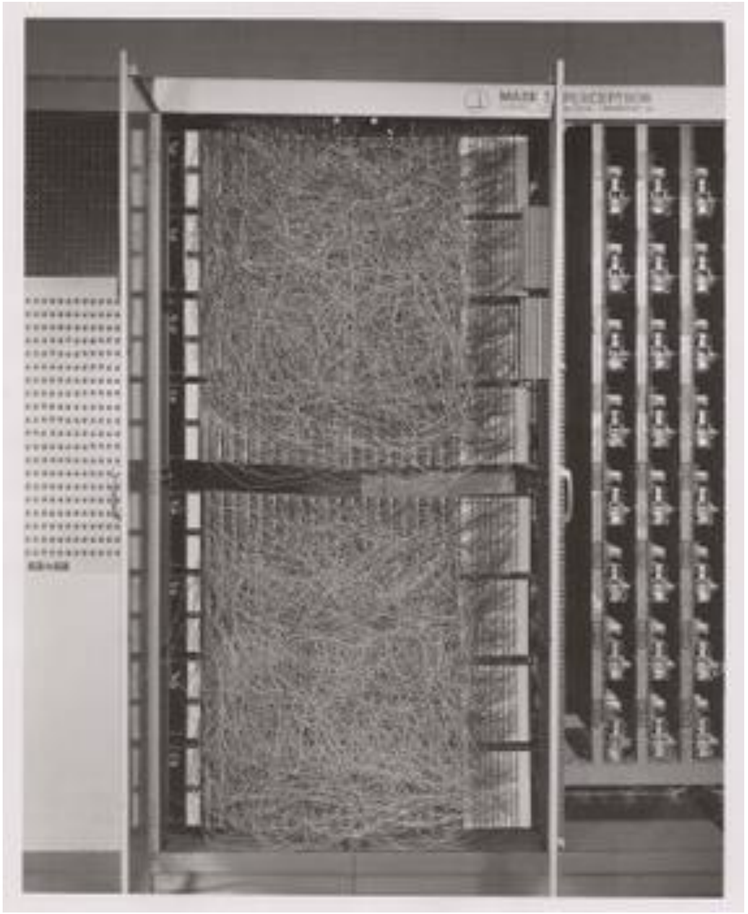
Perceptron Algorithm [Rosenblatt ’57]

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

$$= \frac{1}{n}\sum_{i\in V} -y_i(\mathbf{x}_i^T\mathbf{w})$$

where V 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]

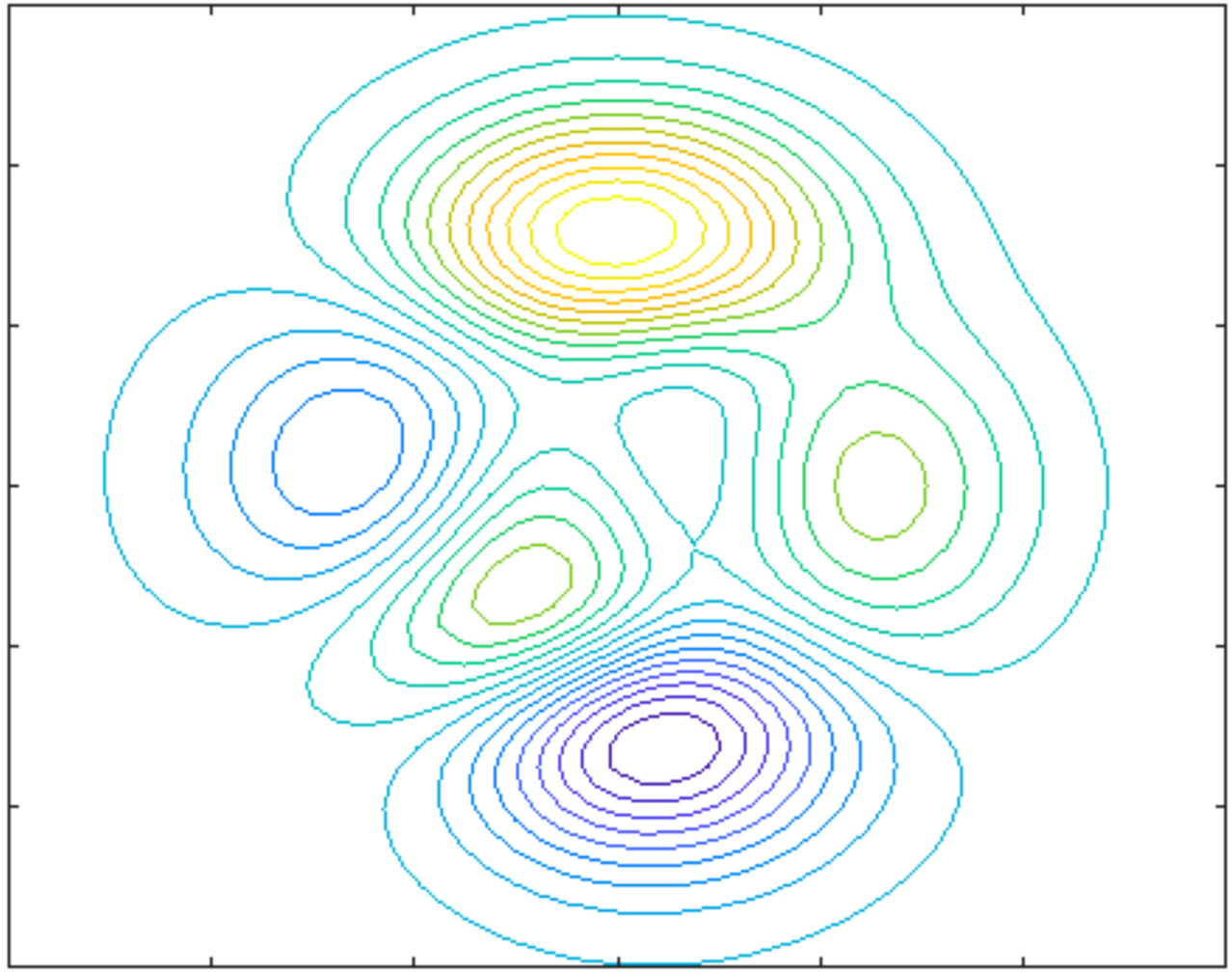
- **Initialize** $\mathbf{w}$
- **While** $\mathscr{L}(\mathbf{w} > 0)$
  - $v \leftarrow$ **set of indices with** $y_i(\mathbf{x}_i^T \mathbf{w}) < 0$
  - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \sum_{i \in V} - y_i \mathbf{x}_i$

Gradient Descent

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

Gradient Descent

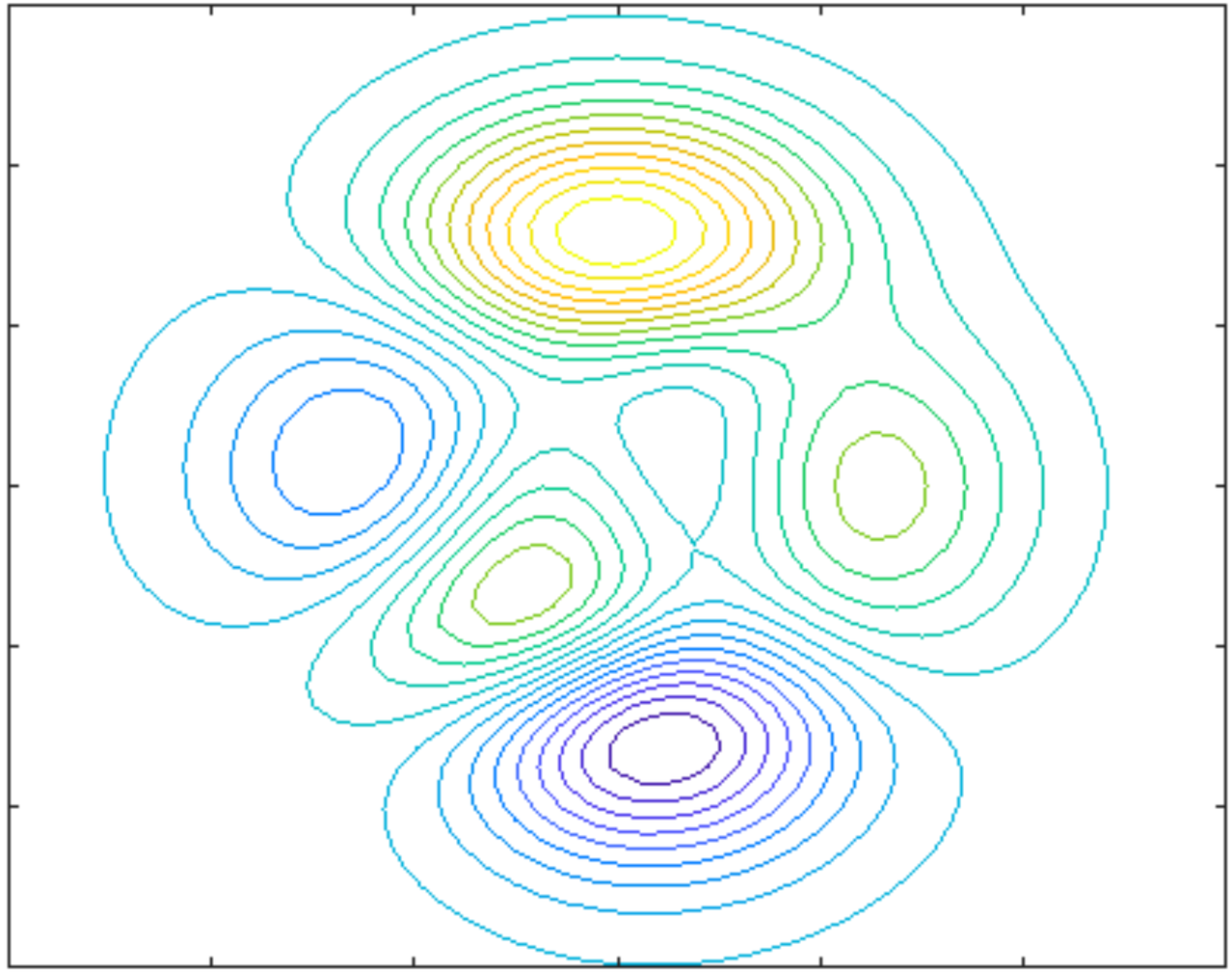
$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$



Gradient Descent

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

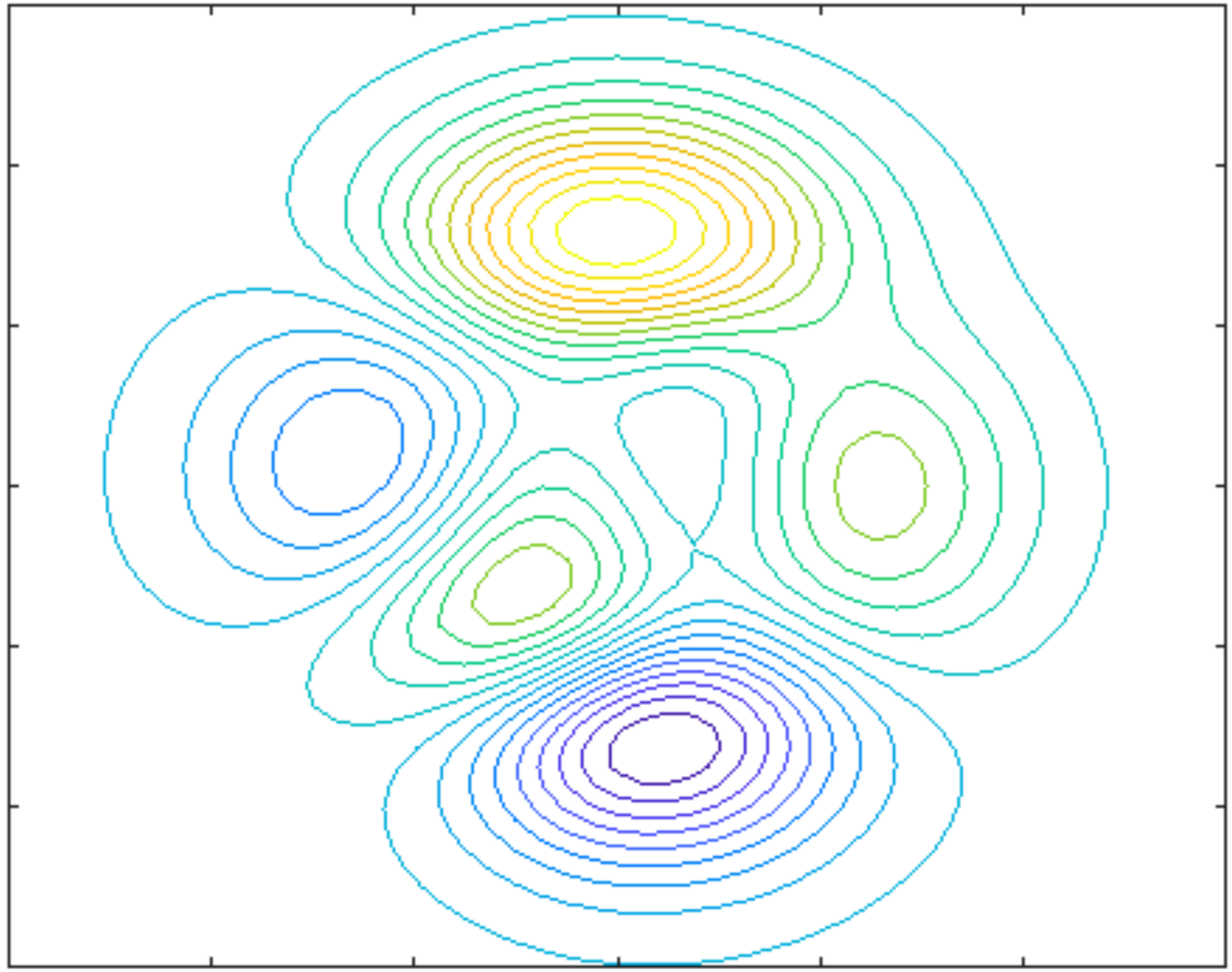
```
w = initialize_weights()
for t in range(num_steps):
  dw = compute_gradient(loss_fn, data, w)
  w -= learning_rate * dw
```



Gradient Descent

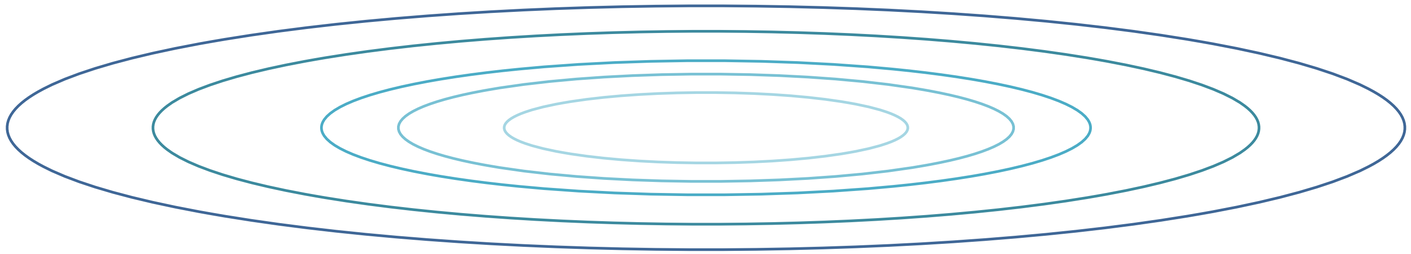
$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

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

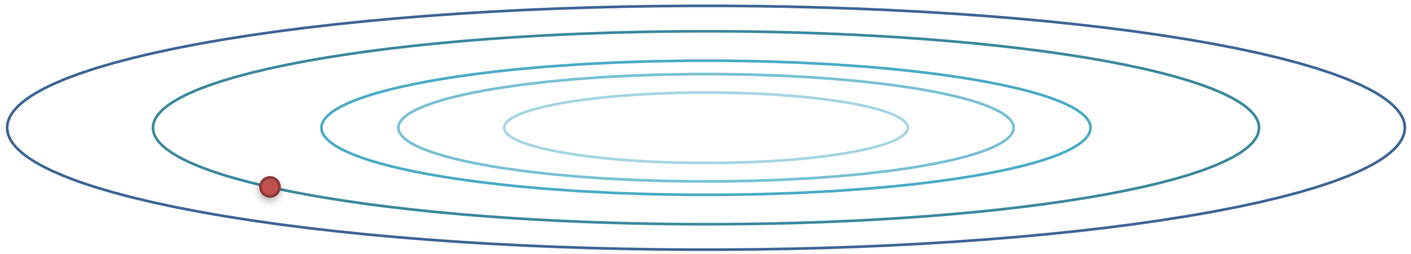


Hyperparameters?
a) intialize_weights
b) learning_rate
c) number of steps/truncation condition

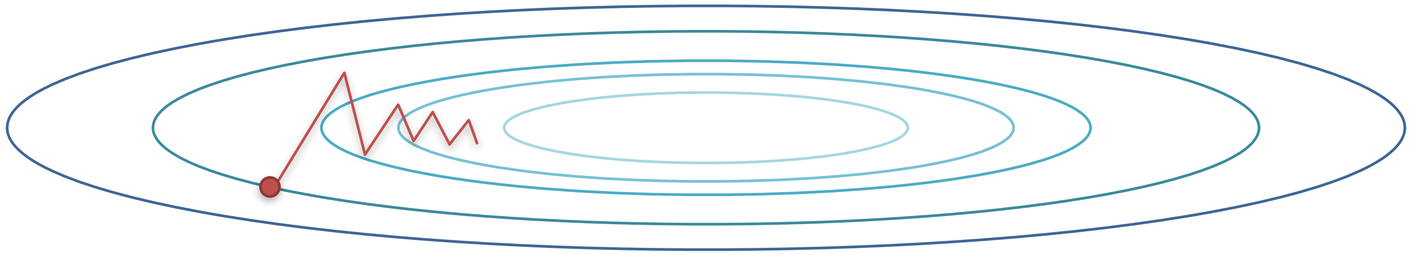
Problems with Gradient Descent



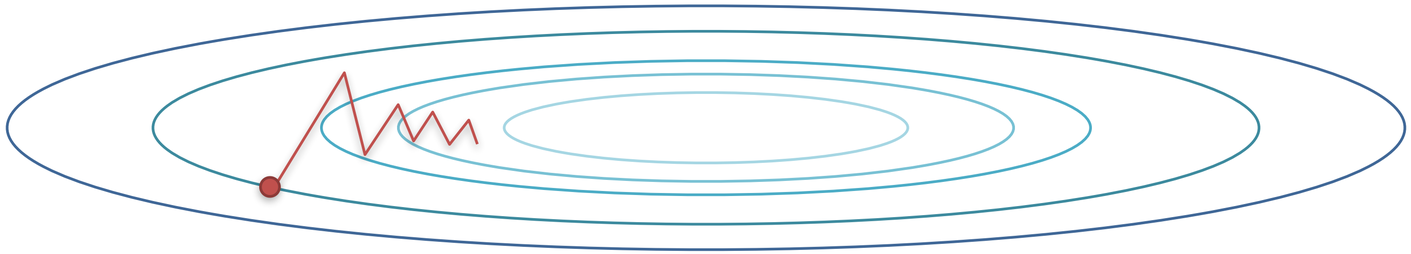
Problems with Gradient Descent



Problems with Gradient Descent

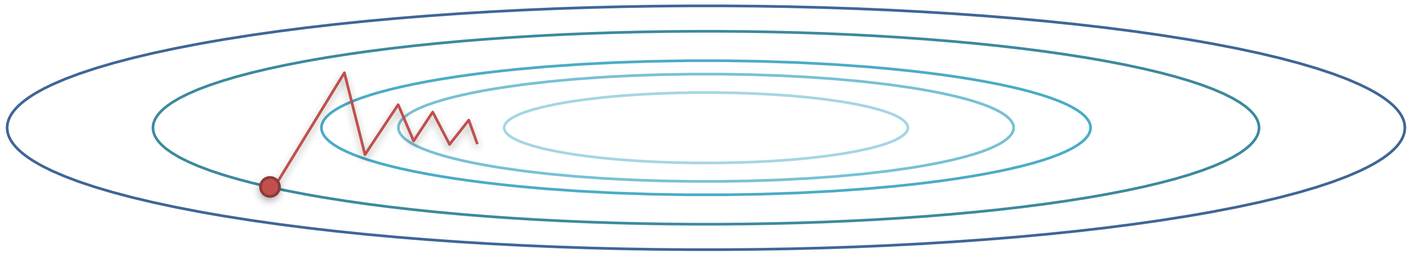


Problems with Gradient Descent



1. Bad condition number → slow convergence

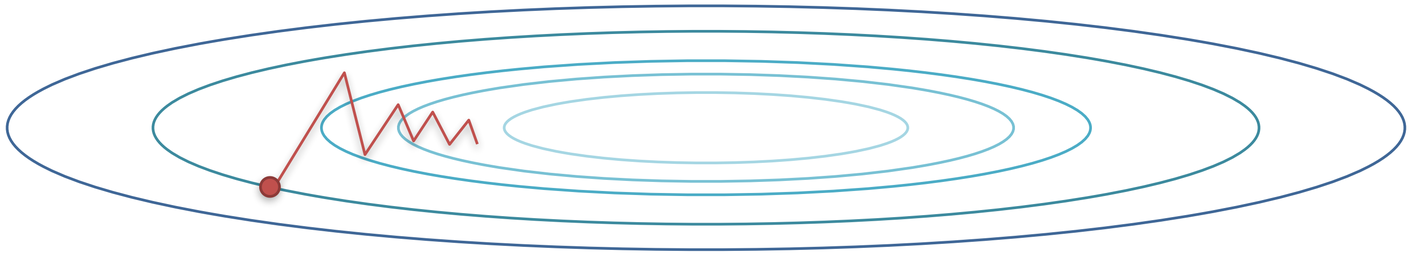
Problems with Gradient Descent



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) intialize_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)

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

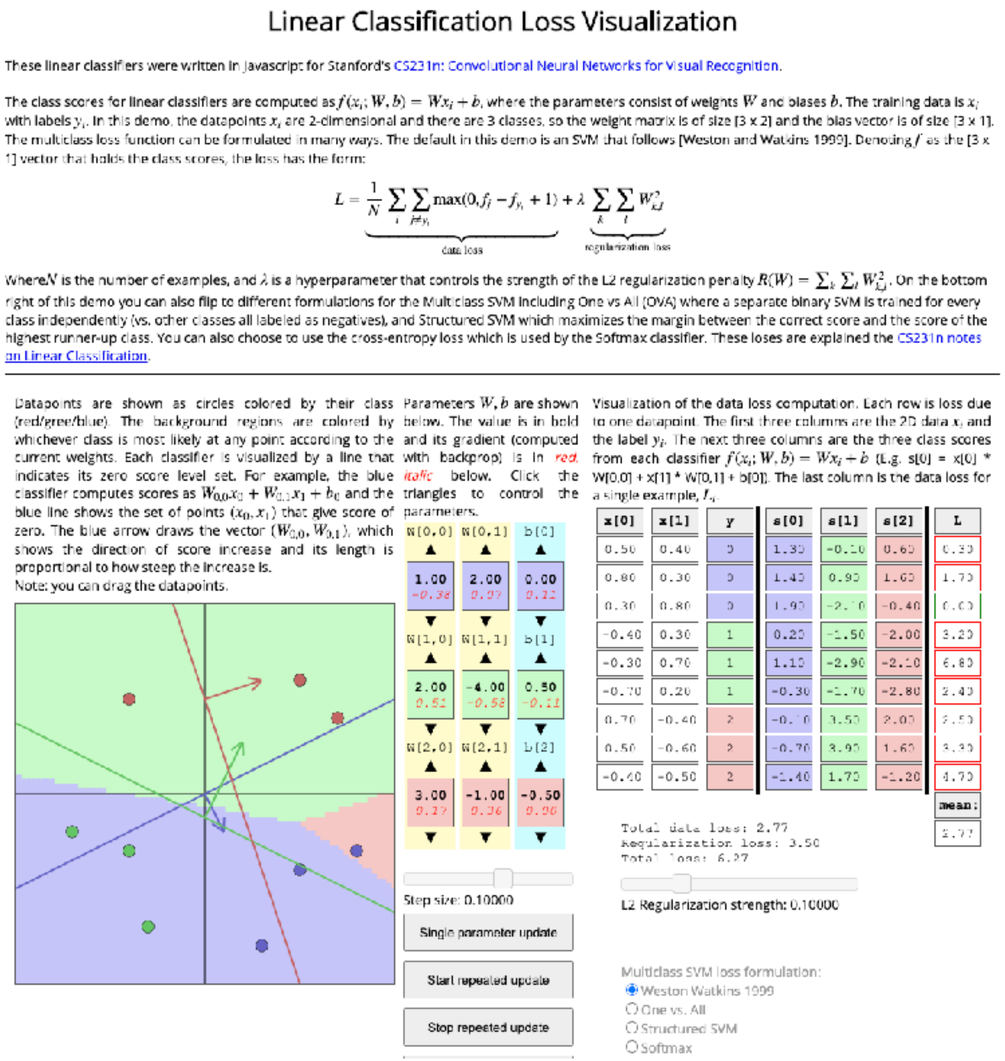
Stochastic Gradient Descent (SGD)

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

```
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
```

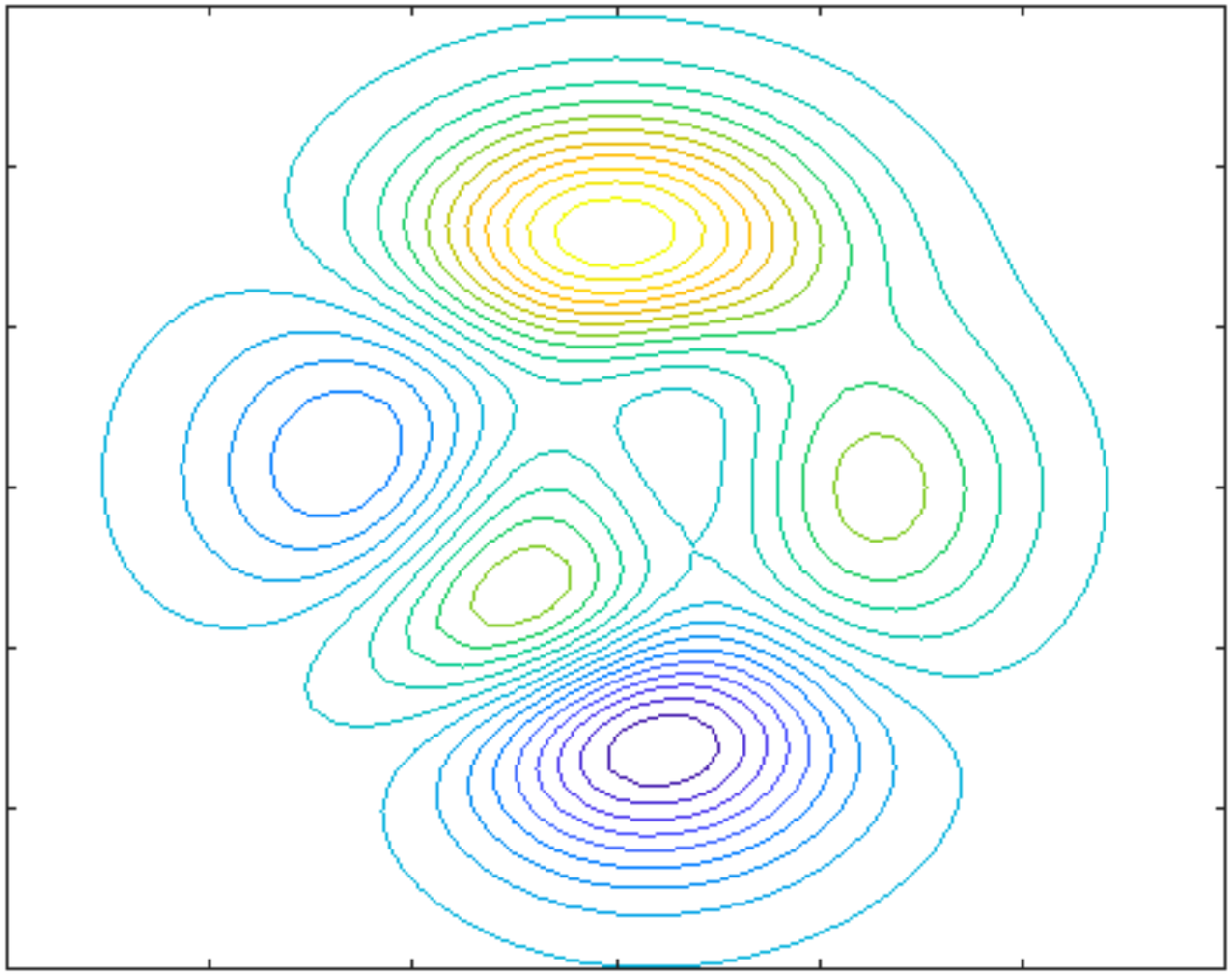
Interactive Demo

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

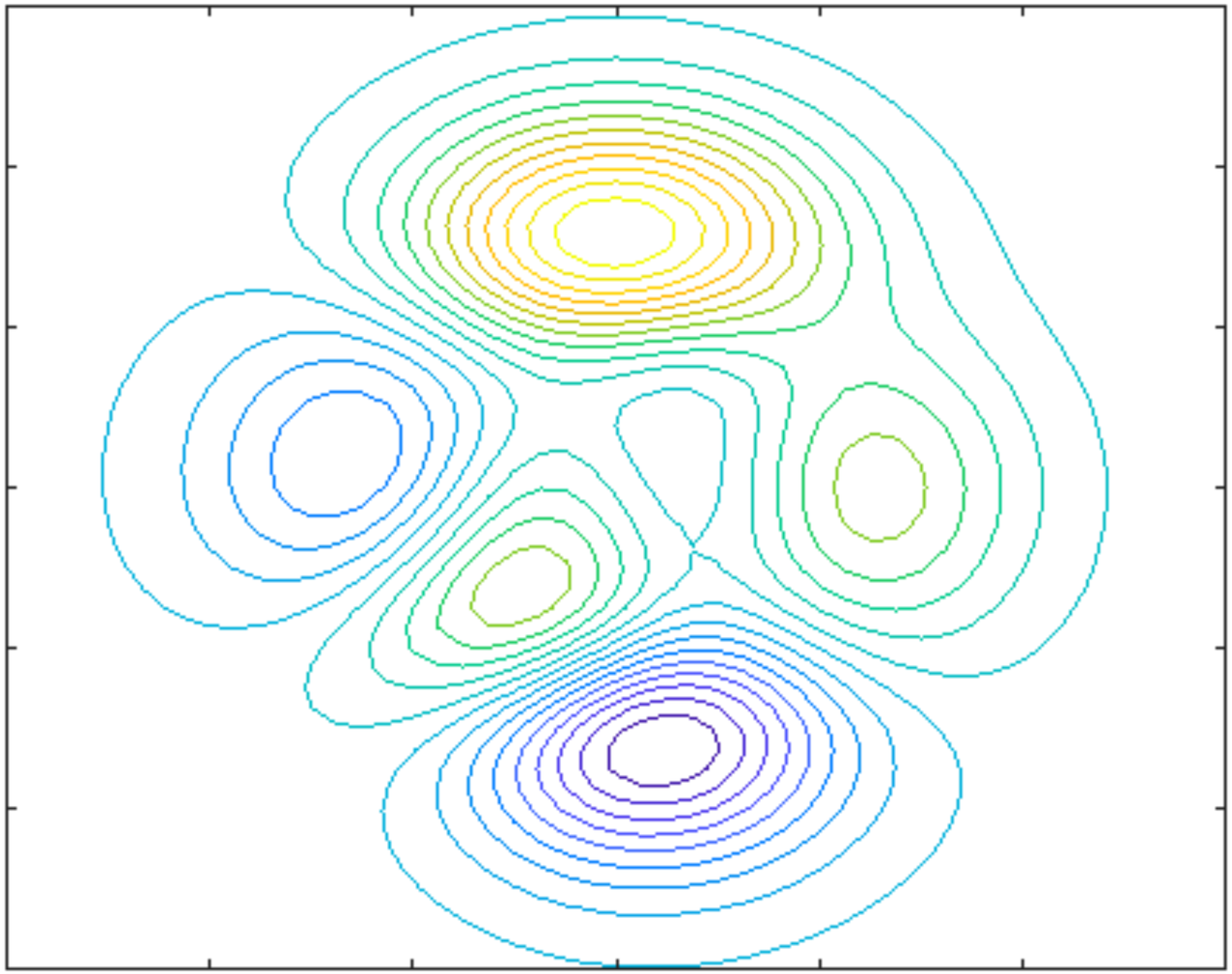


Problems with GD (or SGD)

Problems with GD (or SGD)



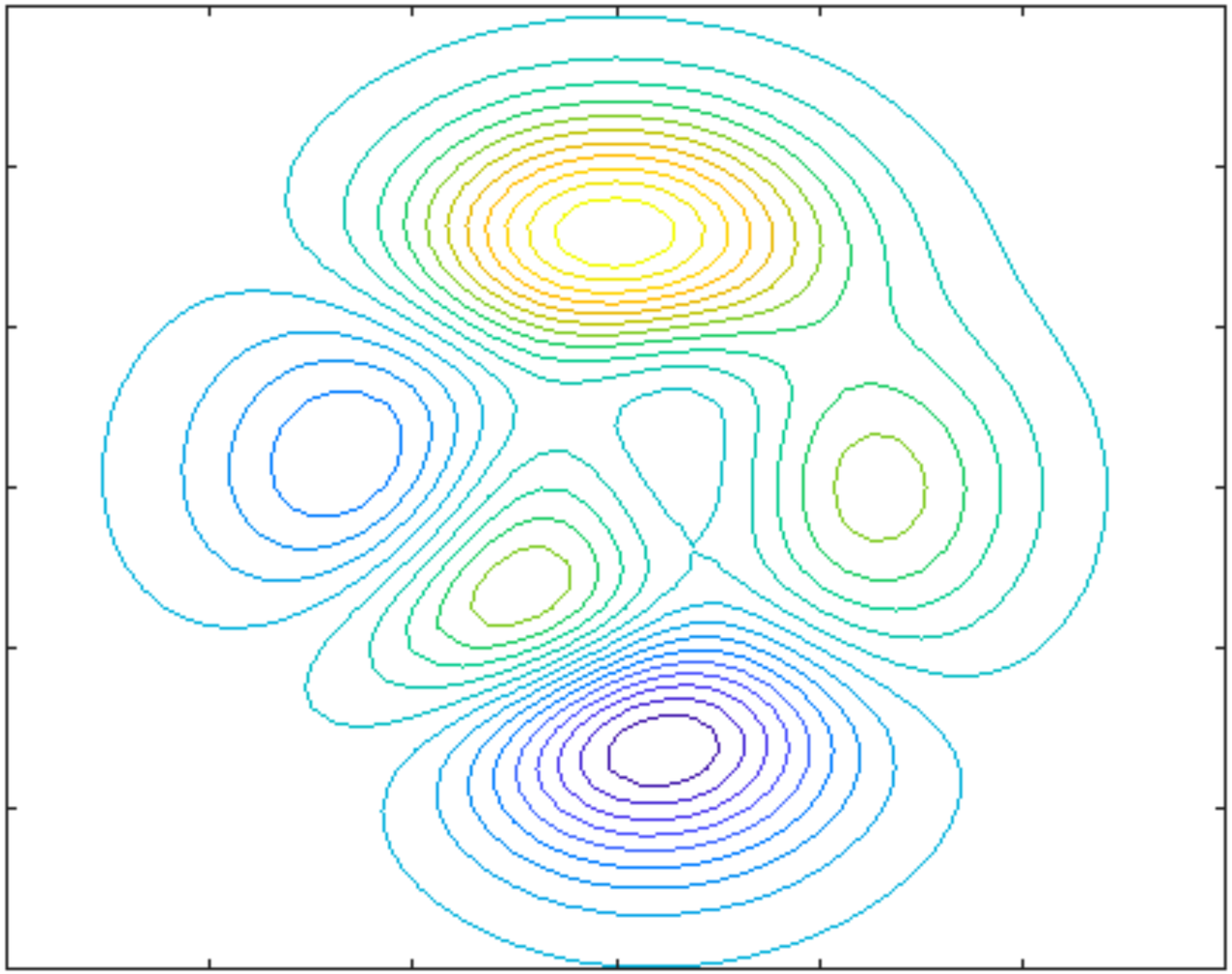
Problems with GD (or SGD)



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

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

Problems with GD (or SGD)

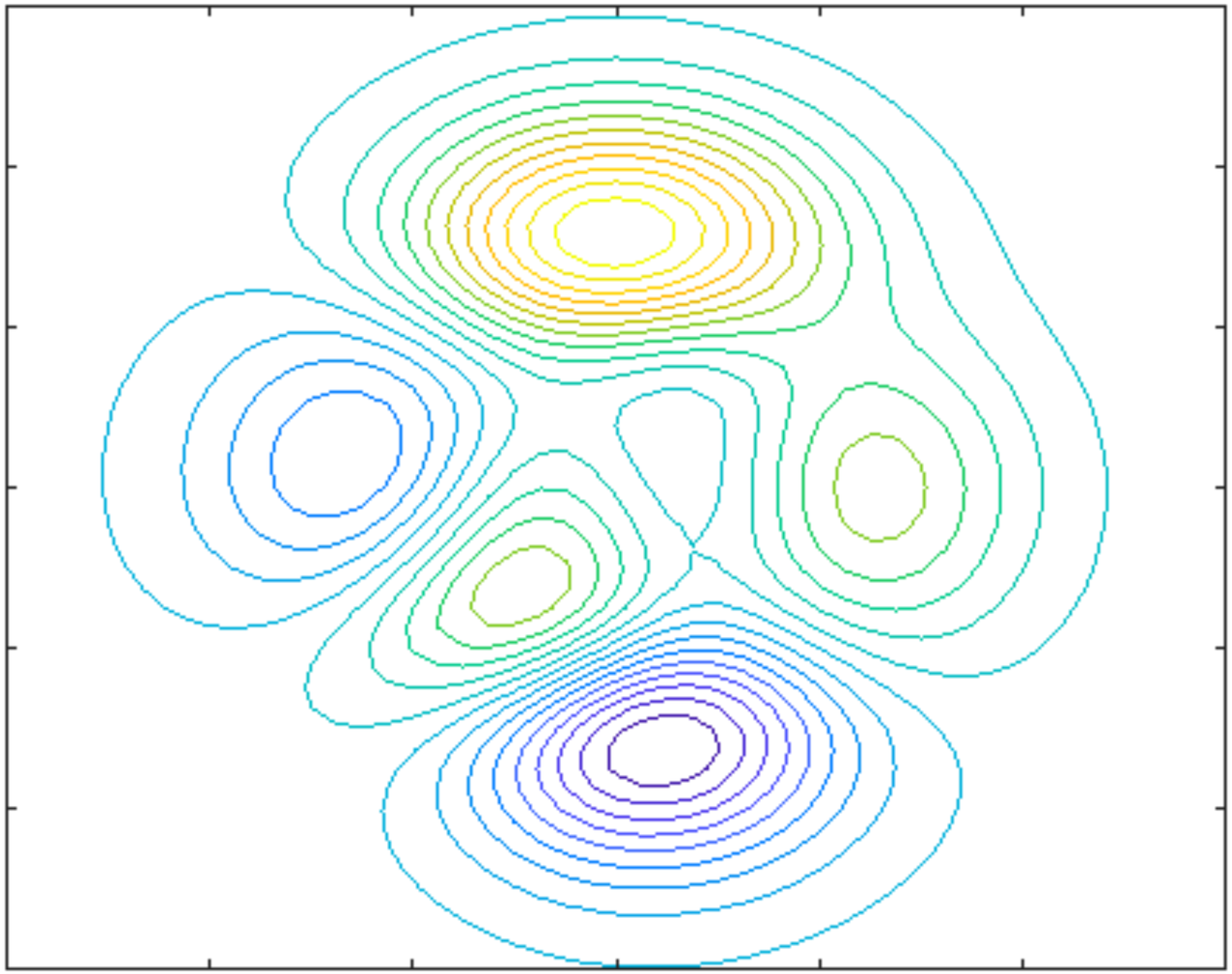


$$v_{t+1} = \rho v_t + \nabla f(w_t)$$

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

$$= w_t - \alpha\rho v_t - \alpha\nabla f(w_t)$$

Problems with GD (or SGD)



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

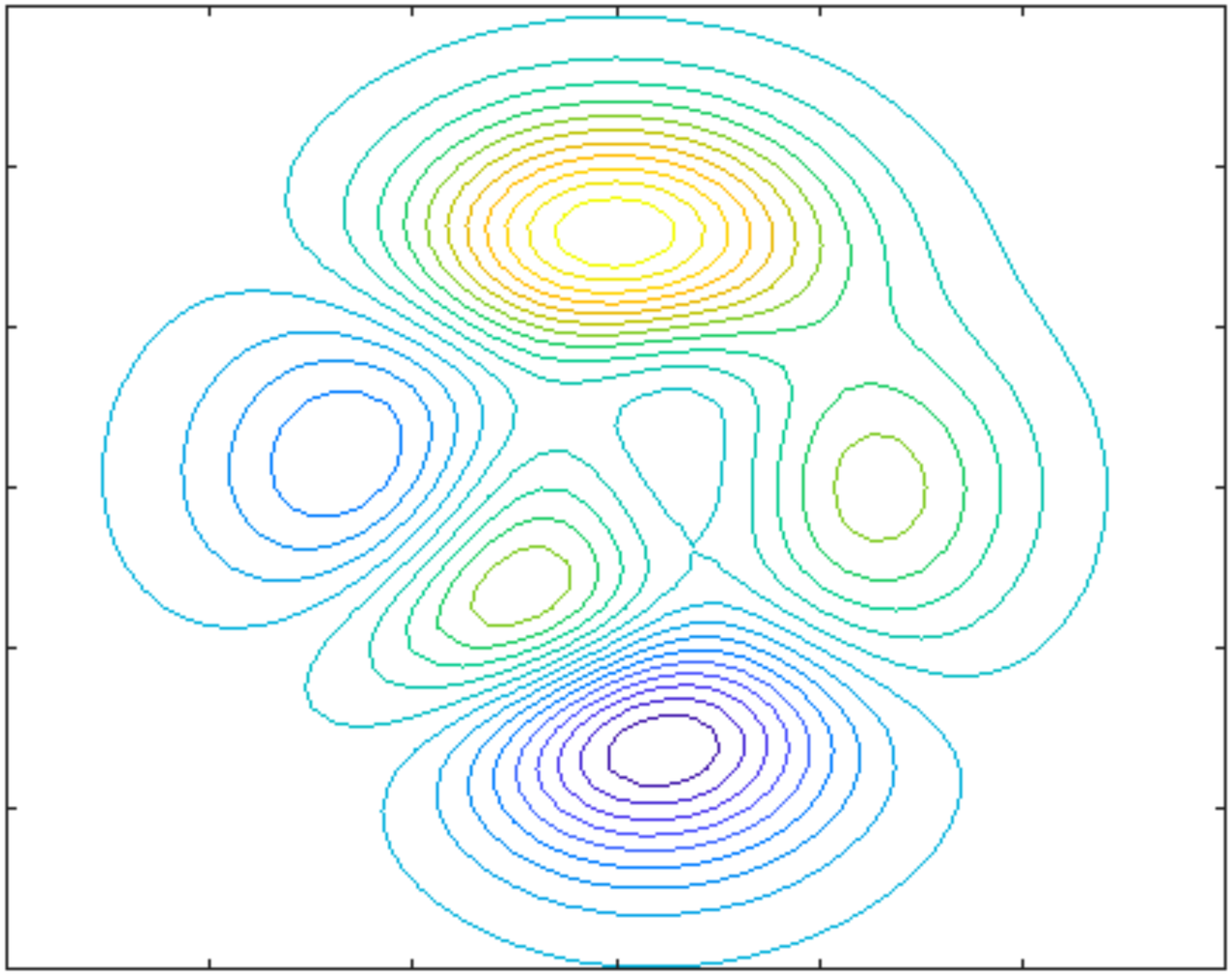
$$w_{t+1} = w_t - \alpha v_{t+1}$$

$$= w_t - \alpha\rho v_t - \alpha\nabla f(w_t)$$

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

$$w_{t+1} = w_t + v_{t+1}$$

Problems with GD (or SGD)



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

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

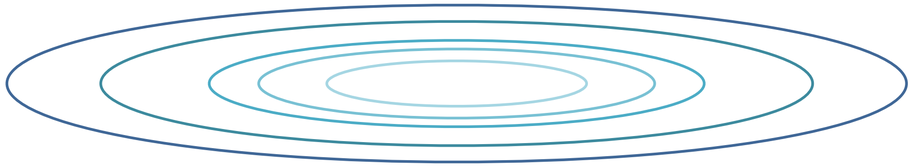
$$= w_t - \alpha\rho v_t - \alpha\nabla f(w_t)$$

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

$$w_{t+1} = w_t + v_{t+1}$$

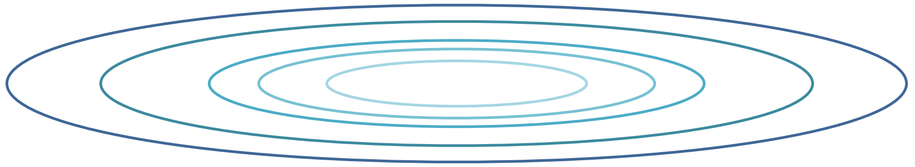
$$= w_t + \rho v_t - \alpha\nabla f(w_t)$$

AdaGrad vs RmsProp



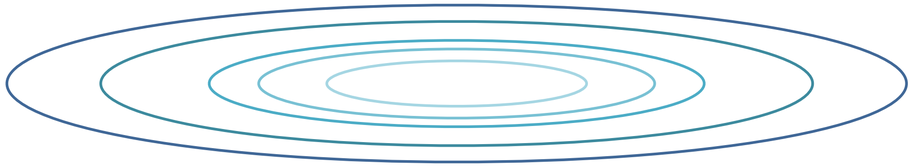
AdaGrad vs RmsProp

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



AdaGrad vs RmsProp

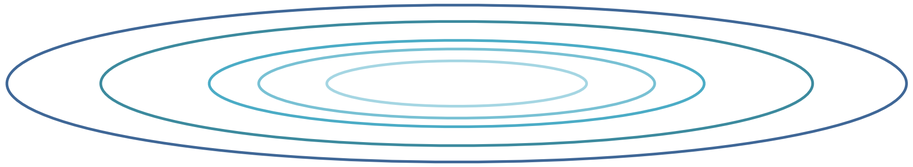
```
w = initialize_weights()
for t in range(num_steps):
  dw = compute_gradient(loss_fn, data, w)
  w -= learning_rate * dw
```



AdaGrad

AdaGrad vs RmsProp

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

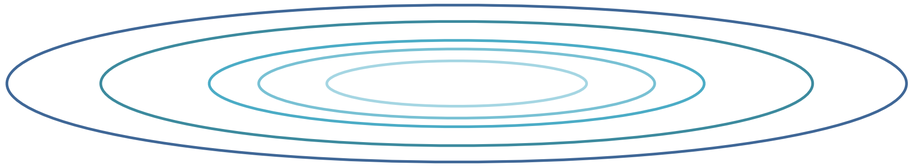


AdaGrad

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

AdaGrad vs RmsProp

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



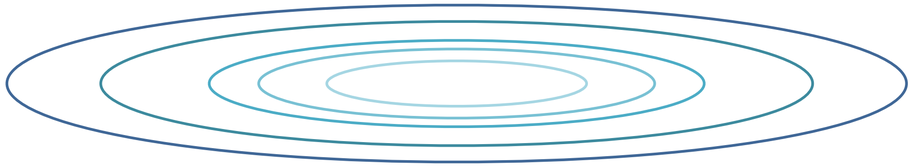
AdaGrad

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

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

AdaGrad vs RmsProp

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



AdaGrad

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

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

RmsProp

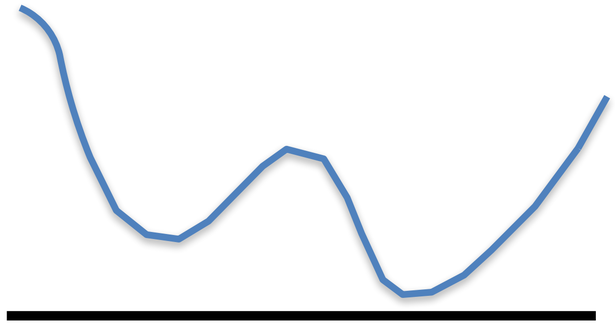
$$\text{gradSquared} = \gamma \text{gradSquared} + (1 - \gamma) \text{dw} * \text{dw}$$

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

Adam (most common choice)

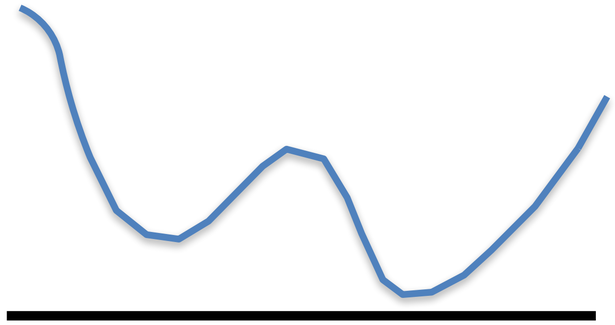
- A bit more than AdaGrad + RmsProp

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



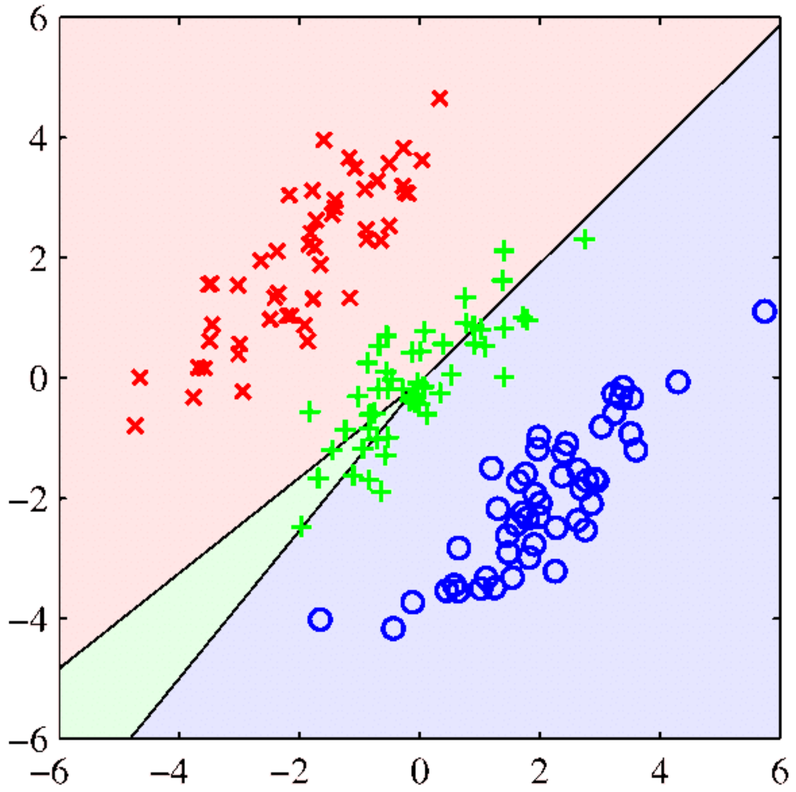
$$F(w) \approx F(w_0) + (w - w_0)^T \nabla_w F(w_0) + \frac{1}{2}(w - w_0)^T H_w (w - w_0)$$

$$w^* = w_0 - \mathbf{H}_\omega^{-1}(\omega_0) \nabla_\omega \mathbf{F}(\omega_0)$$

Logistic vs Linear Regression, n>2 classes

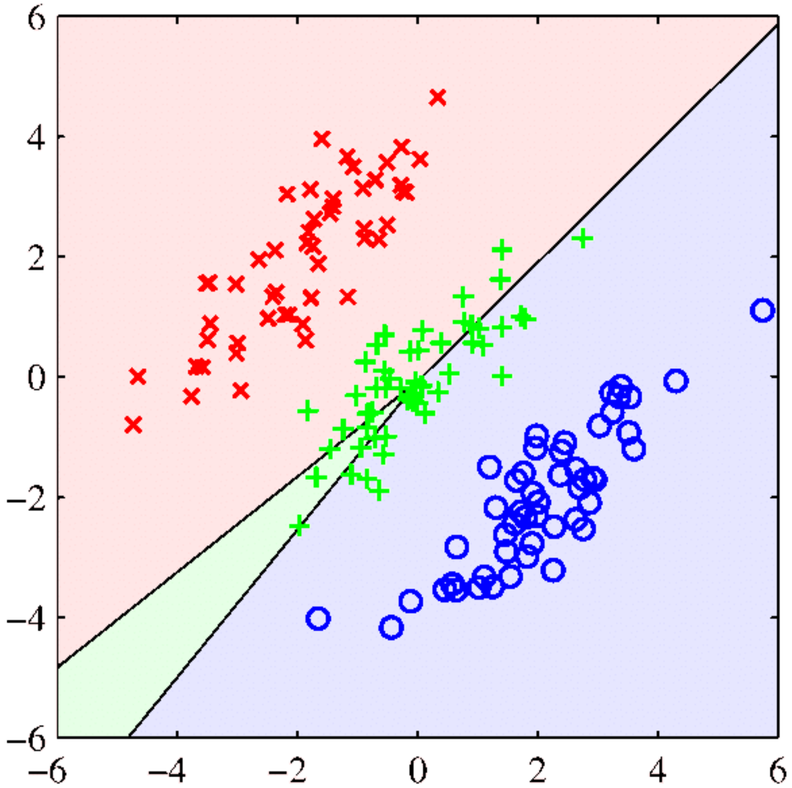
Logistic vs Linear Regression, n>2 classes

Linear regression

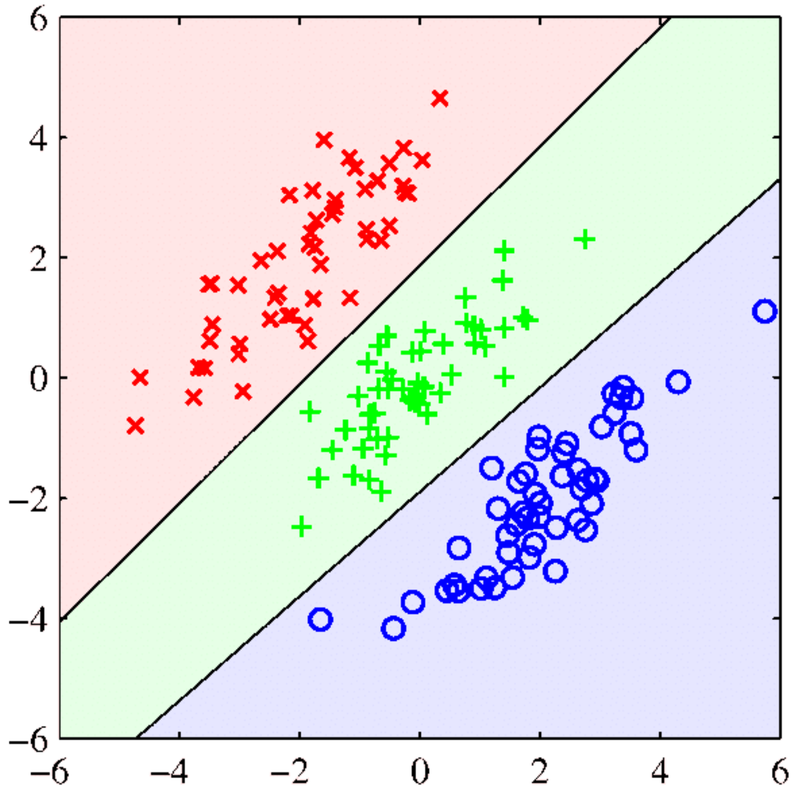


Logistic vs Linear Regression, n>2 classes

Linear regression

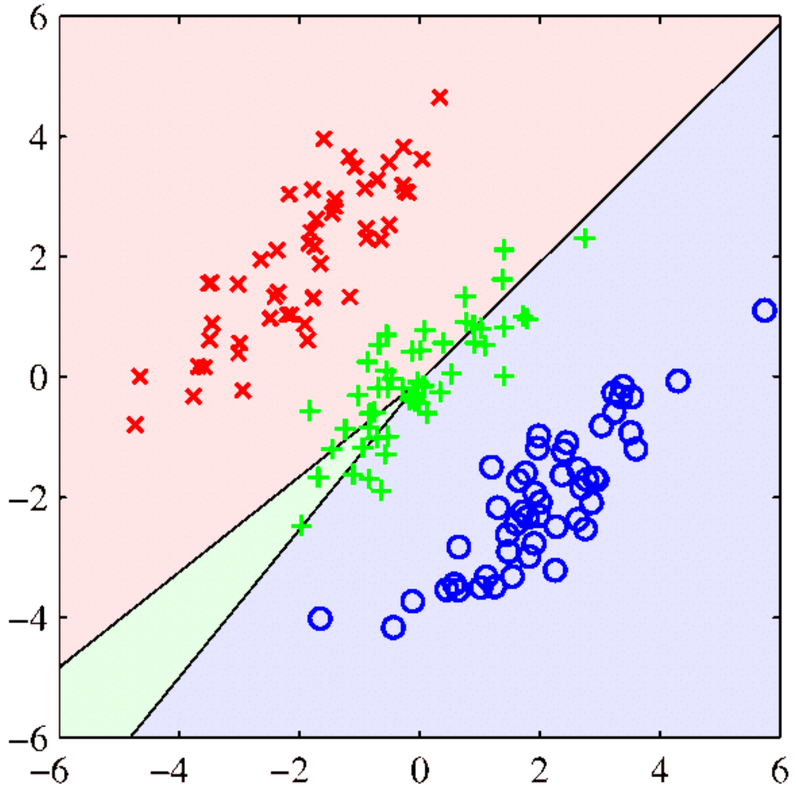


Logistic regression

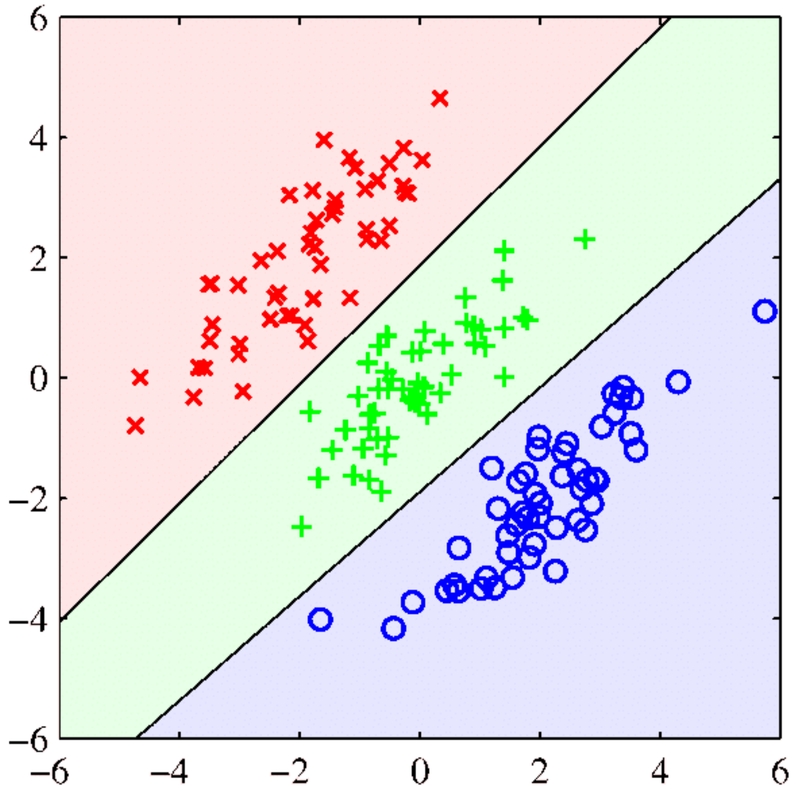


Logistic vs Linear Regression, n>2 classes

Linear regression



Logistic regression



Logistic regression does not exhibit the masking problem

Form of posterior distribution

Bernoulli-type conditional distribution

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

Particular choice of form of f:

Form of posterior distribution

Bernoulli-type conditional distribution

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

Particular choice of form of f:

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

Form of posterior distribution

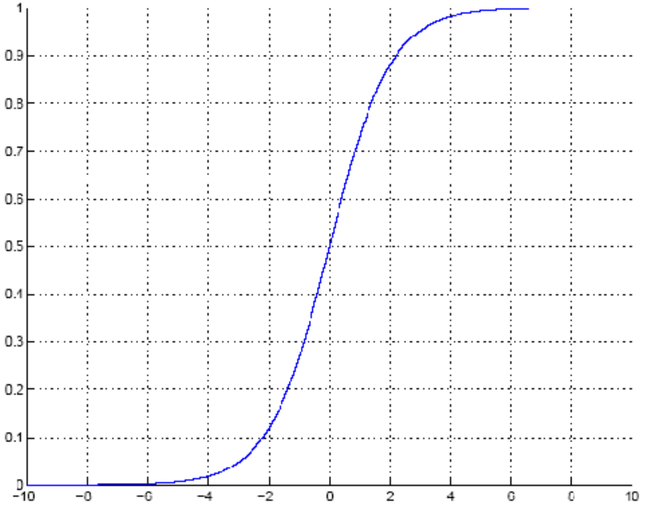
Bernoulli-type conditional distribution

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

Particular choice of form of f:

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

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



Form of posterior distribution

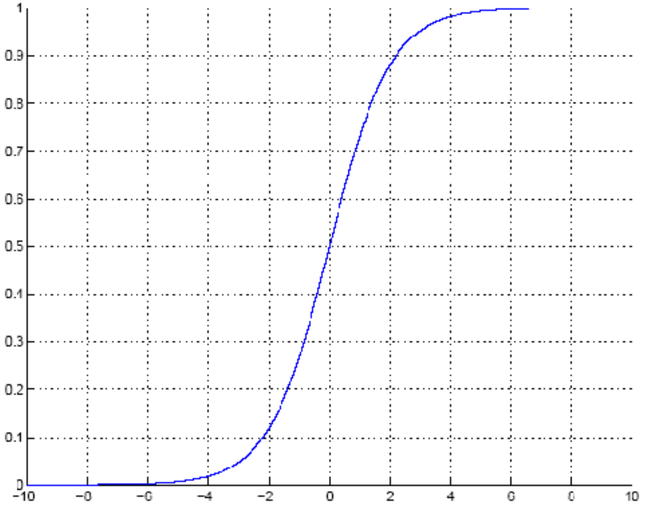
Bernoulli-type conditional distribution

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

Particular choice of form of f:

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

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



"squashing function":

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

Form of posterior distribution

Bernoulli-type conditional distribution

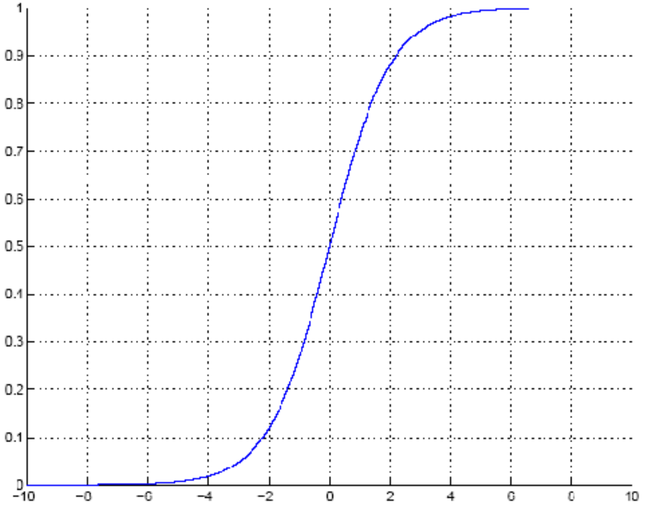
$$\left.\begin{aligned} P(Y=1|X=\mathbf{x};\mathbf{w}) &= f(\mathbf{x},\mathbf{w}) \\ P(Y=0|X=\mathbf{x};\mathbf{w}) &= 1-f(\mathbf{x},\mathbf{w}) \end{aligned}\right\} \rightarrow$$

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

Particular choice of form of f:

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

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



"squashing function":

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

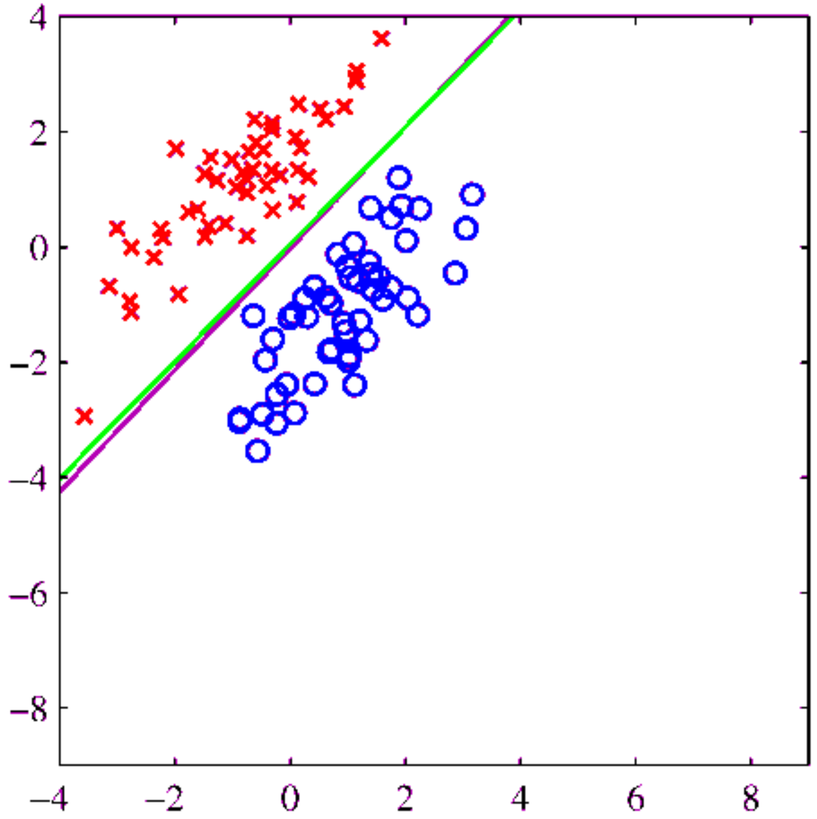
Negative LogLikelihood

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

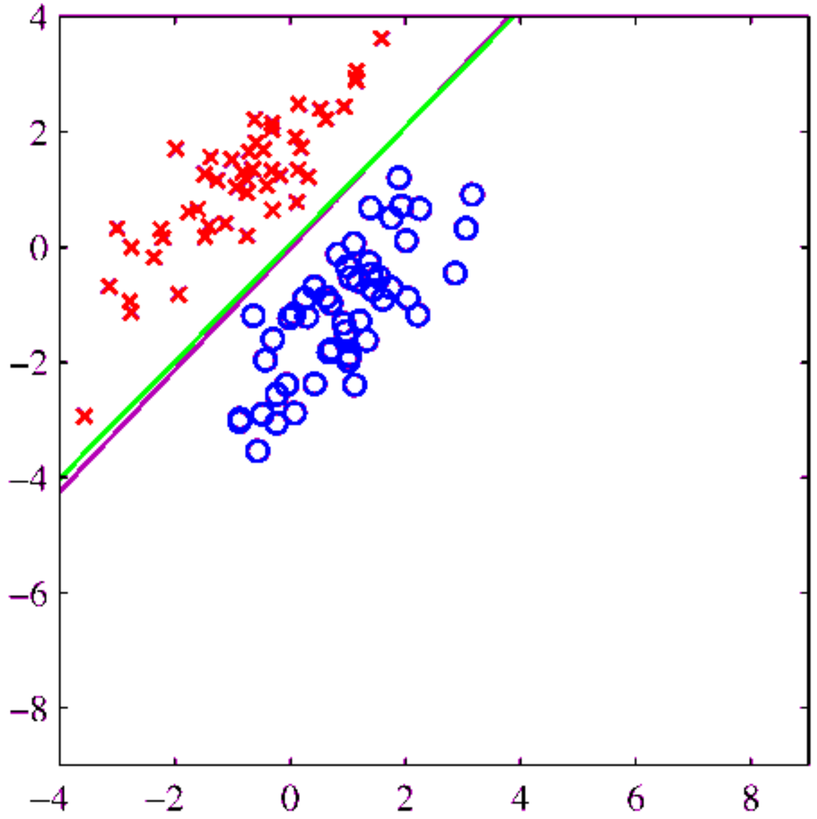
Negative LogLikelihood

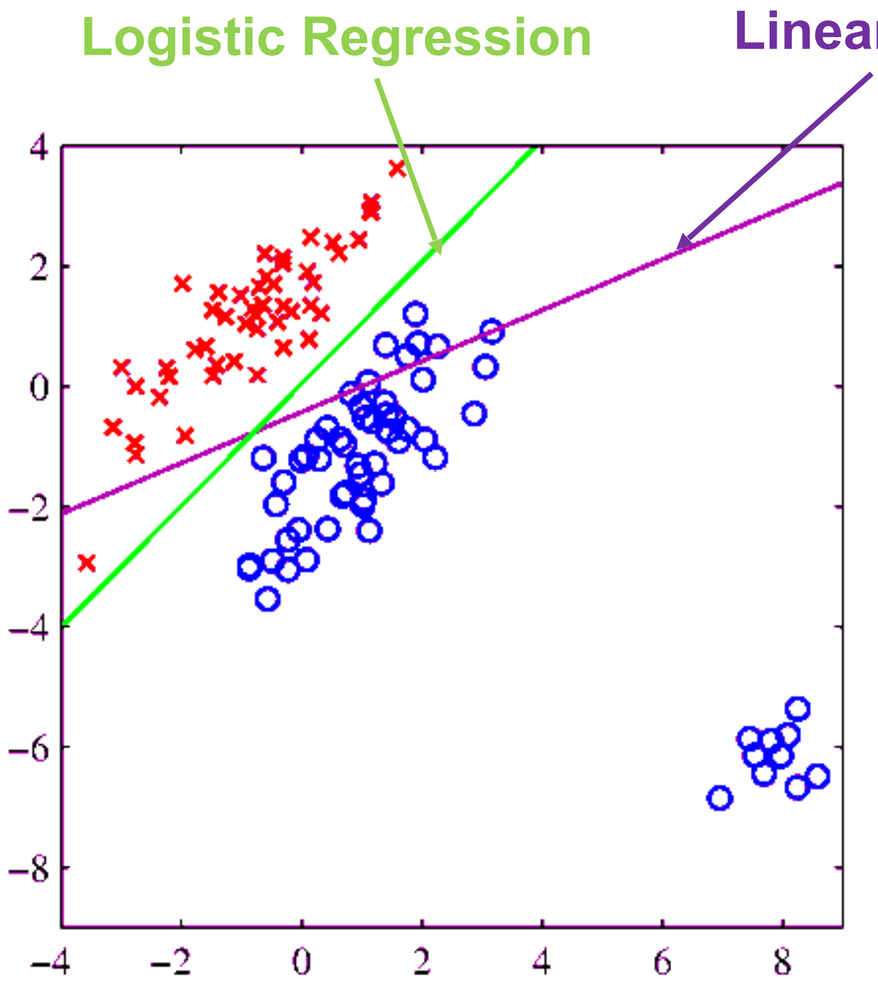
$$\log \mathbb{P}(\mathbf{D}|\theta) = \sum_i (y_i \log \hat{y}_{\theta,i} + (1 - y_i) \log(1 - \hat{y}_{\theta,i}))$$

Logistic vs Linear Regression



Logistic vs Linear Regression





From Two to Many

- How about multi-class classification?

Multiple Classes & Linear Regression

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

Multiple Classes & Linear Regression

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

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

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

Multiple Classes & Linear Regression

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

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

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

Matrix notation:

Multiple Classes & Linear Regression

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

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

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

Matrix notation:

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

Multiple Classes & Linear Regression

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

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

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

Matrix notation:

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

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

Multiple Classes & Linear Regression

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

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

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

Matrix notation:

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

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

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

Multiple Classes & Linear Regression

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

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

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

Matrix notation:

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

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

Loss function:

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

Multiple Classes & Linear Regression

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

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

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

Matrix notation:

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

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

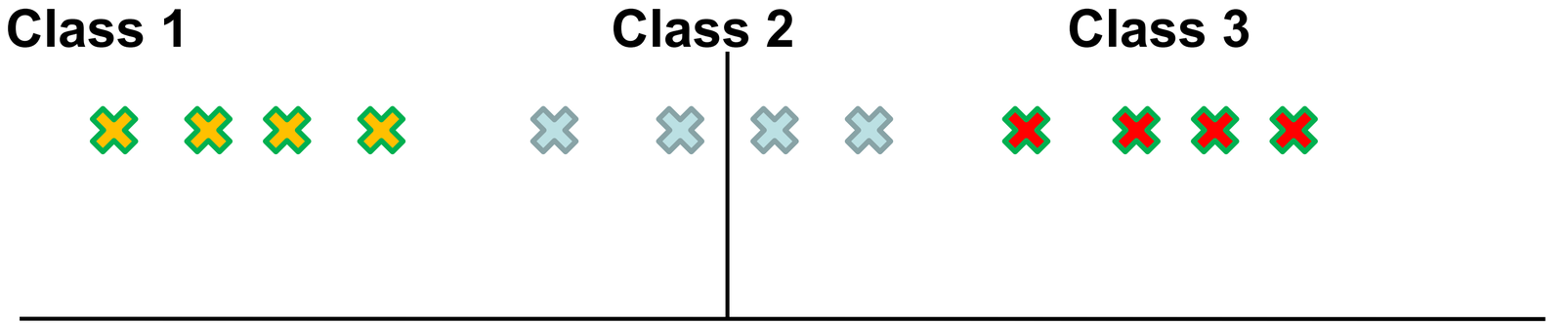
Loss function:

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

Least squares fit (decouples per class):

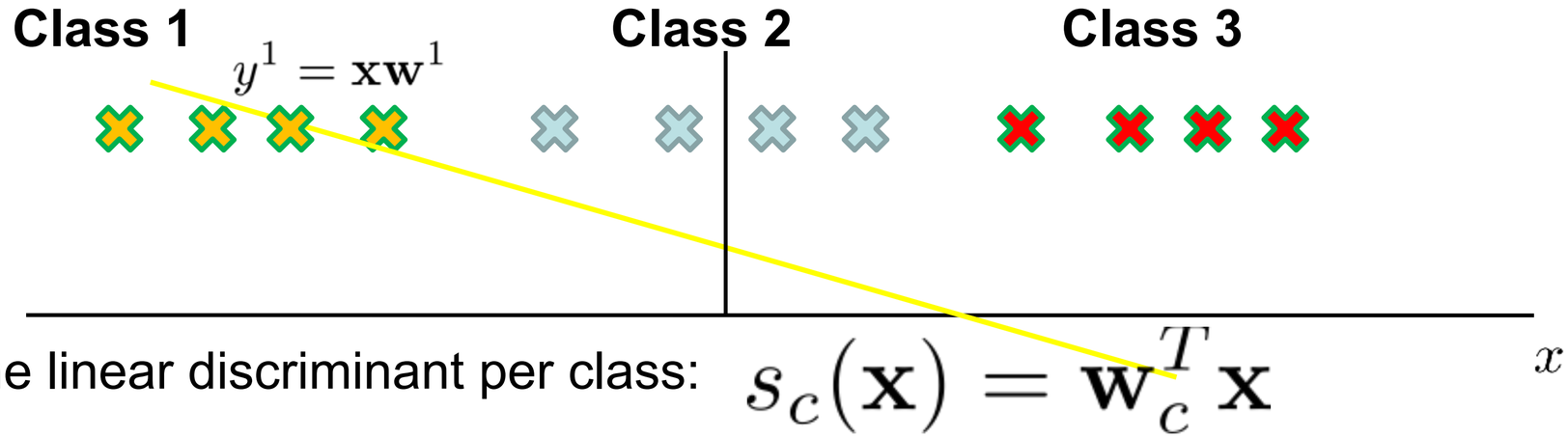
$$\mathbf{w}_c^* = (\mathbf{X}^T\mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}_c$$

Linear Regression Masking Problem



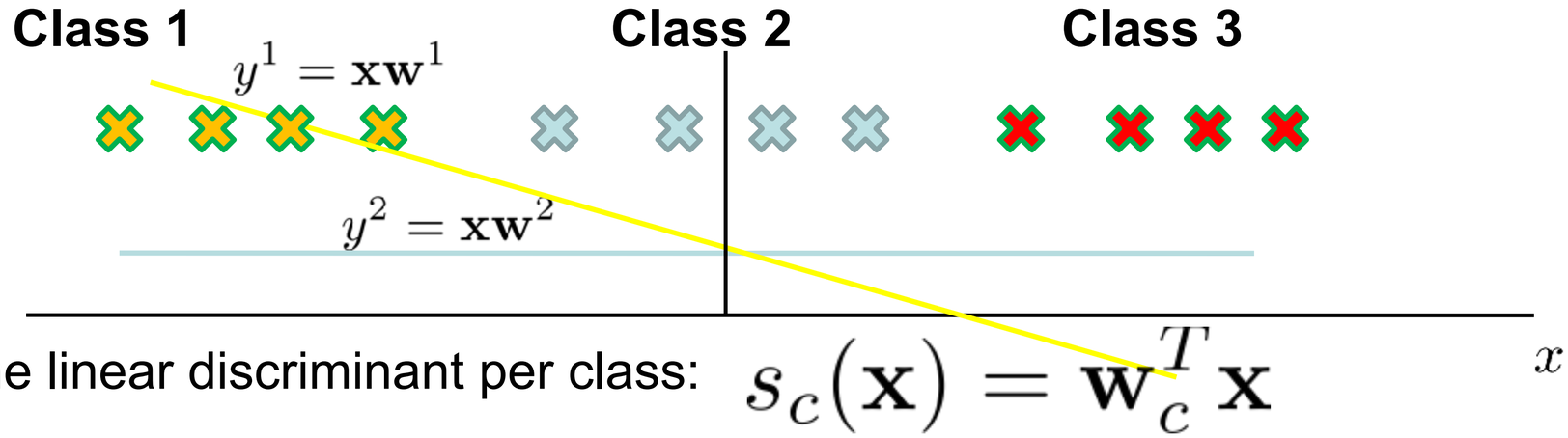
One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

Linear Regression Masking Problem



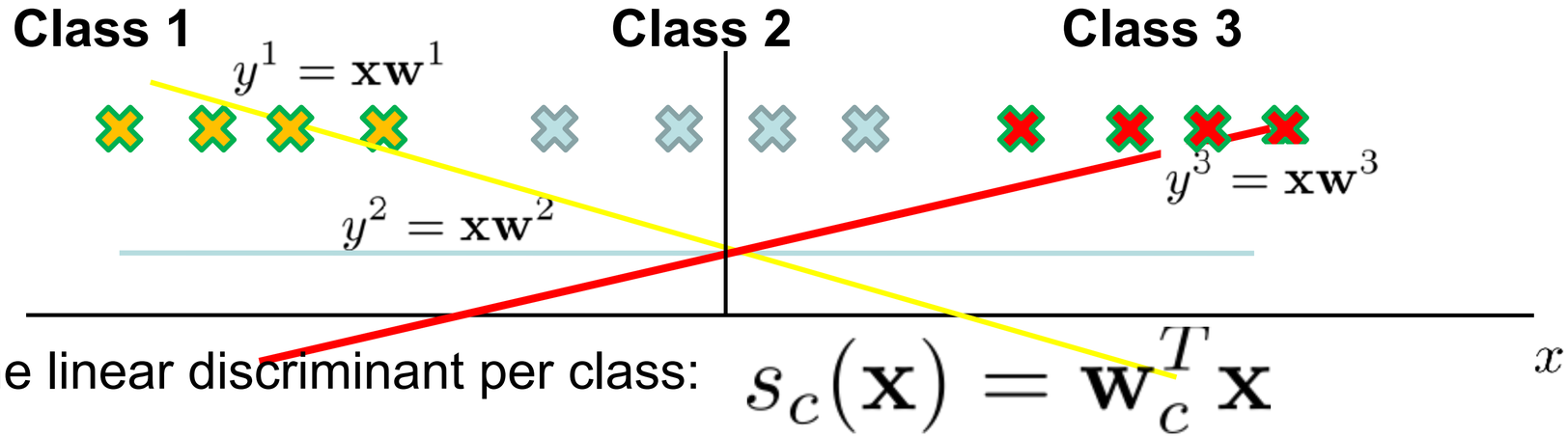
One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

Linear Regression Masking Problem



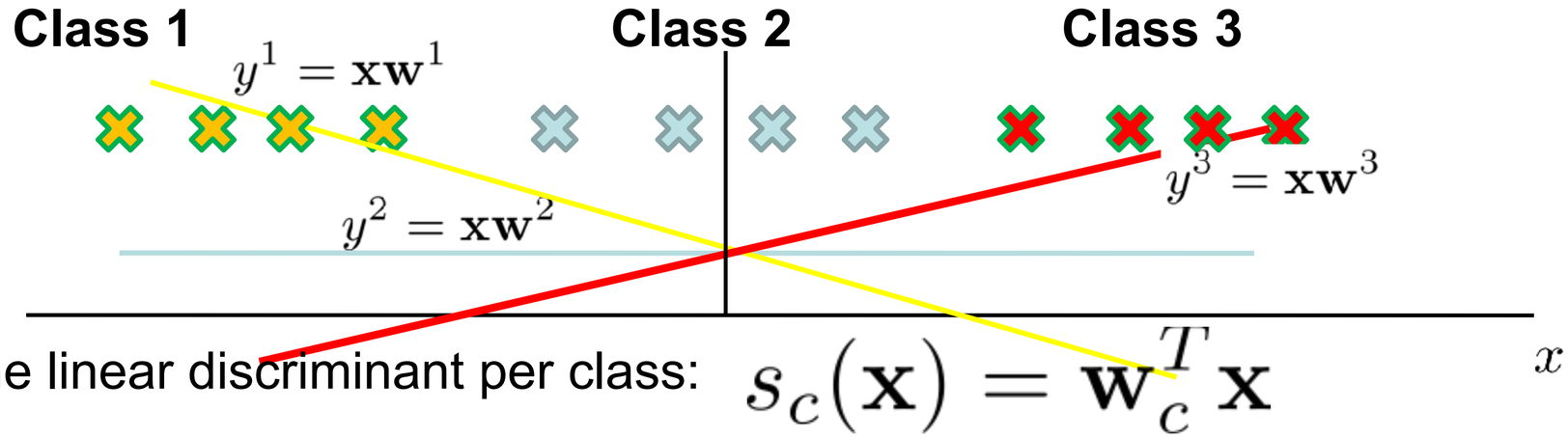
One linear discriminant per class: $s_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x}$

Linear Regression Masking Problem



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

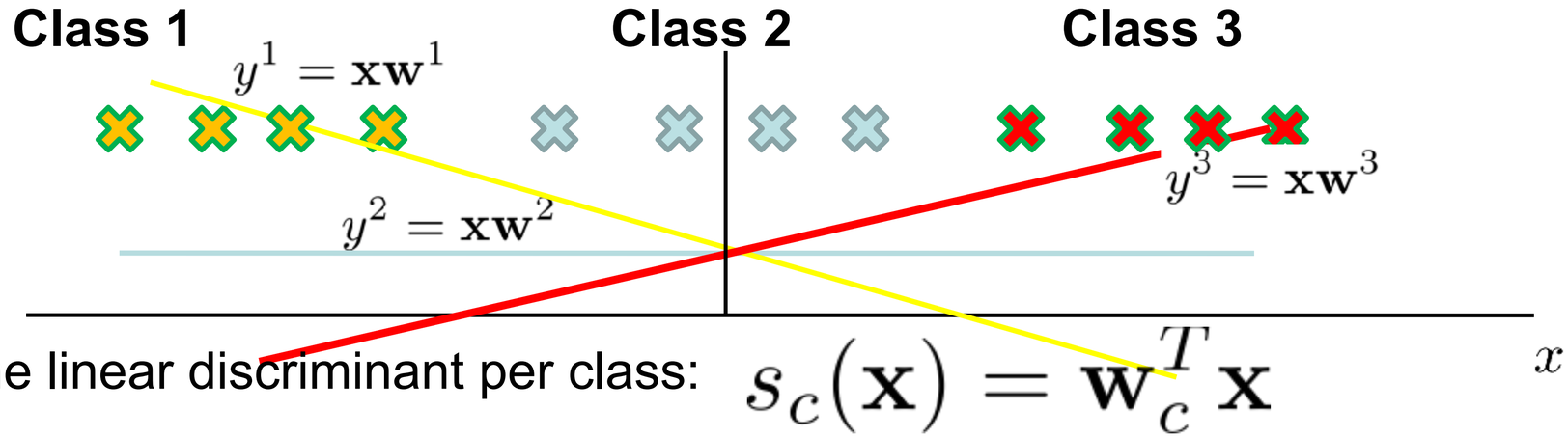
Linear Regression Masking Problem



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

Nothing ever gets assigned to class 2!

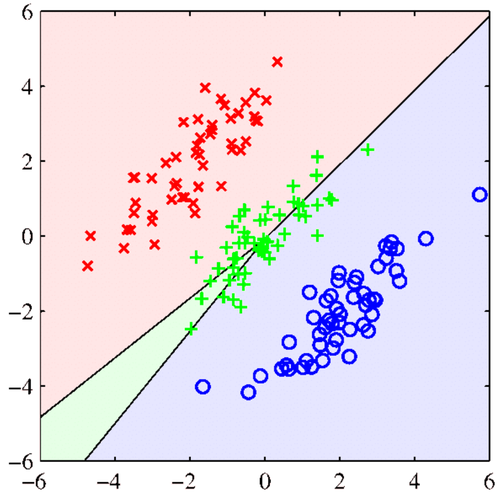
Linear Regression Masking Problem



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

Nothing ever gets assigned to class 2!

2D version:



Multiple classes & Logistic regression

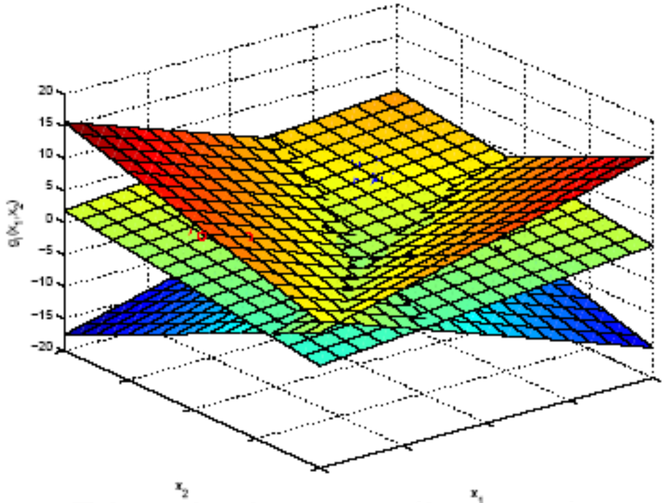
Soft maximum (softmax) of competing classes:

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

Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

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

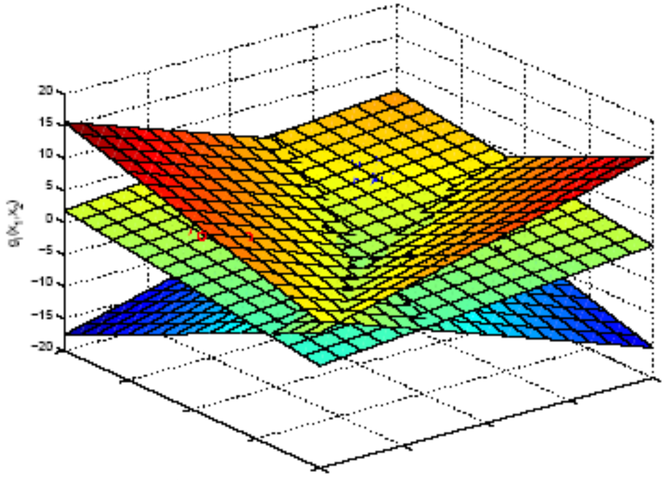


Discriminants (inputs)

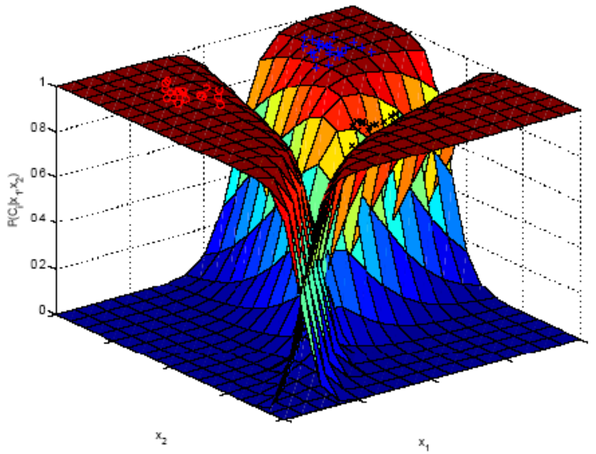
Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

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

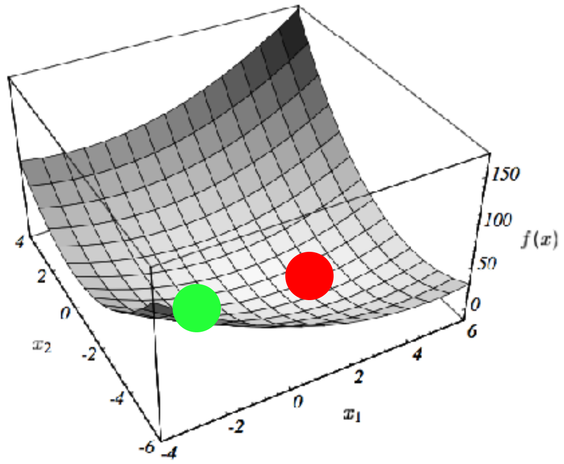


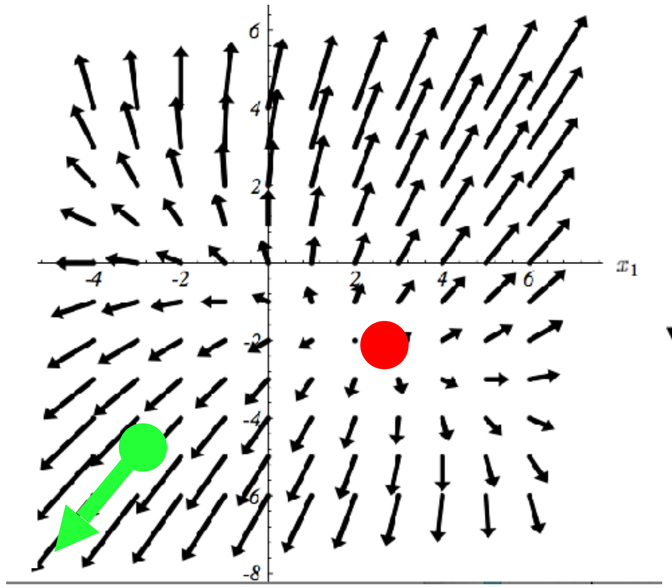
Discriminants (inputs)



Softmax (outputs)

Gradient Descent Minimization

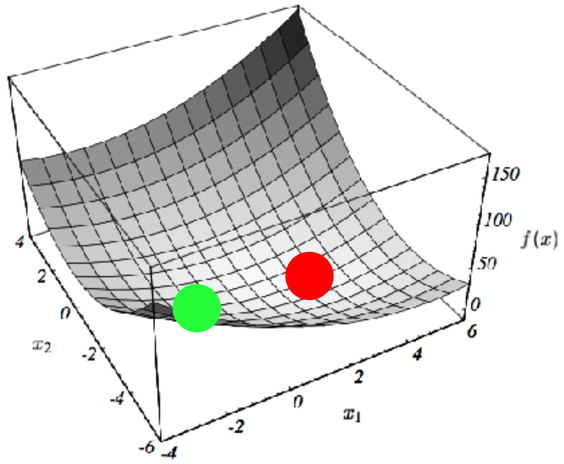


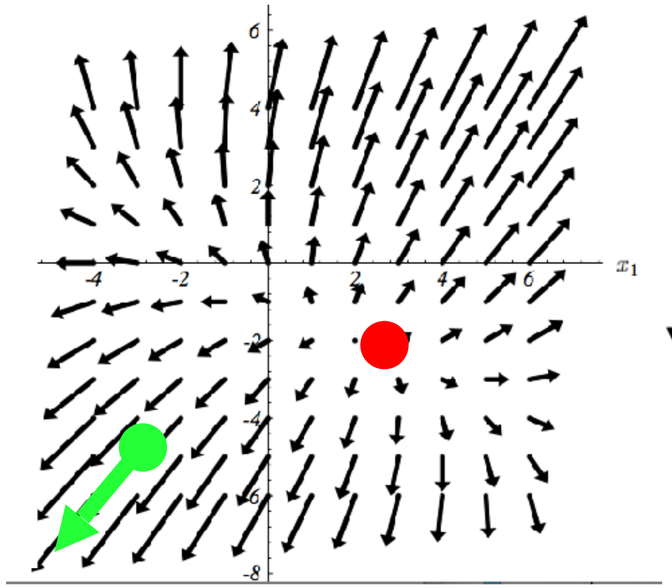


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

Gradient Descent Minimization



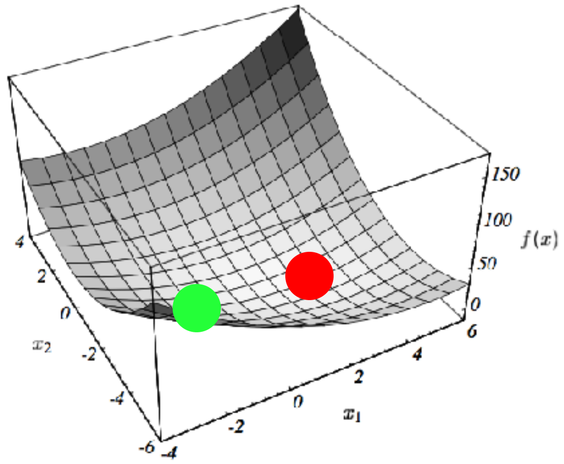


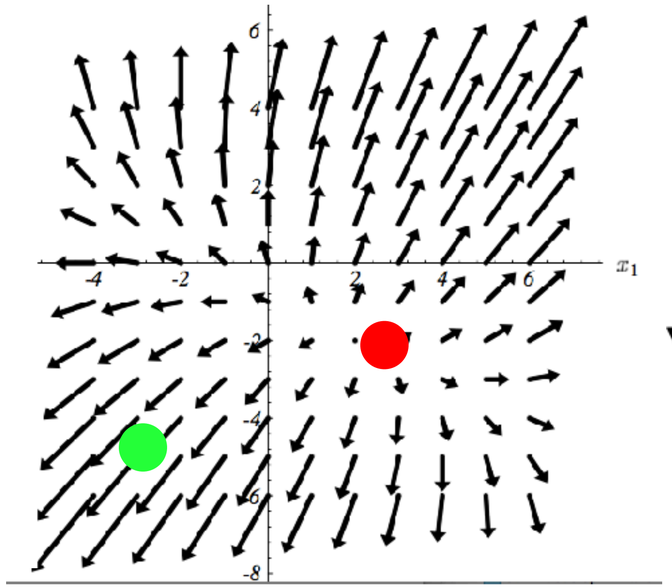
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

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

Gradient Descent Minimization



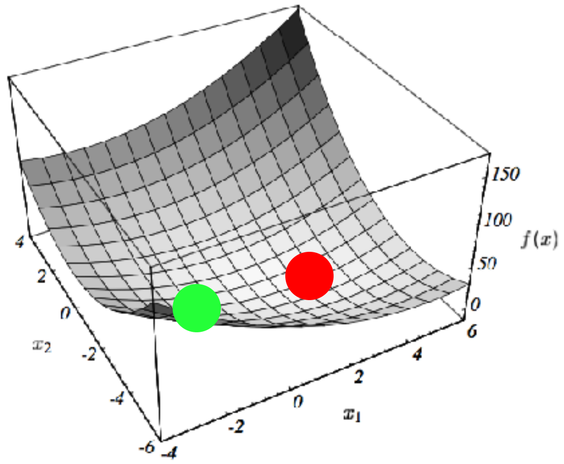


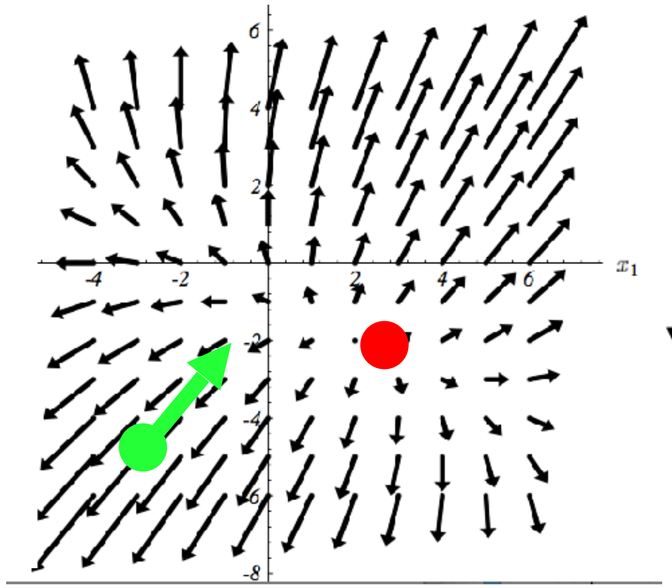
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

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

Gradient Descent Minimization



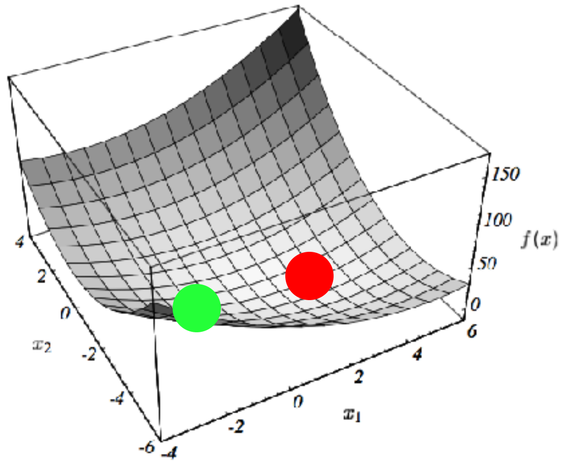


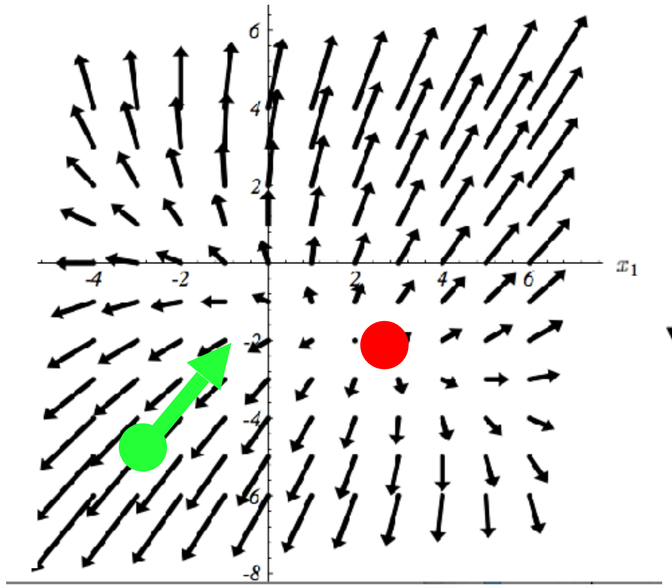
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

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

Gradient Descent Minimization





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

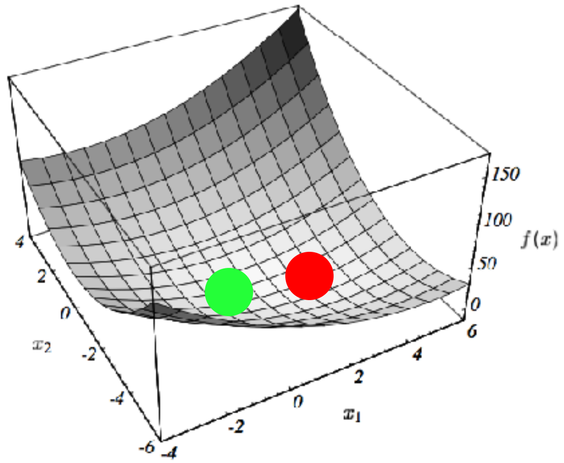
Fact: gradient at any point gives direction of fastest increase

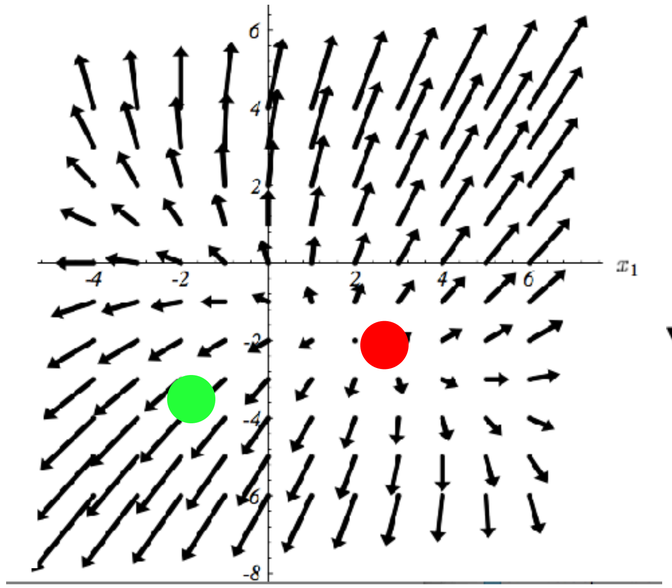
Idea: start at a point and move in the direction opposite to the gradient

Initialize: $\mathbf{x}_0$

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

Gradient Descent Minimization

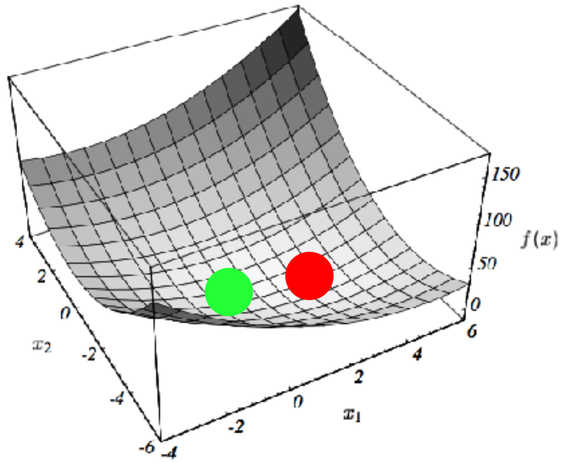


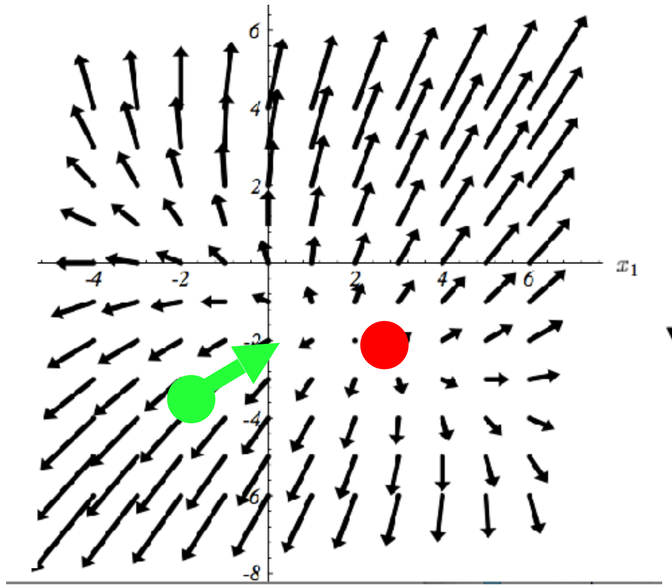


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient Descent Minimization

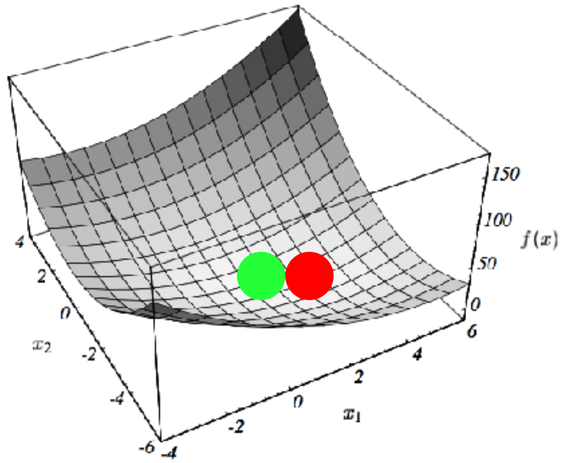


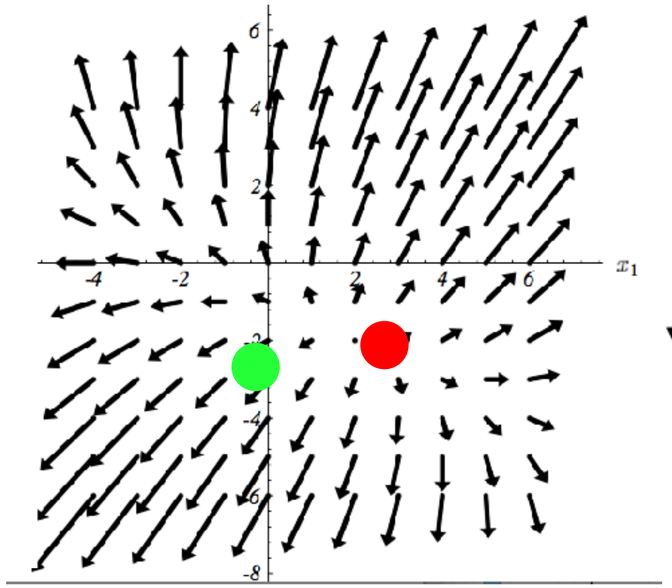


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient Descent Minimization

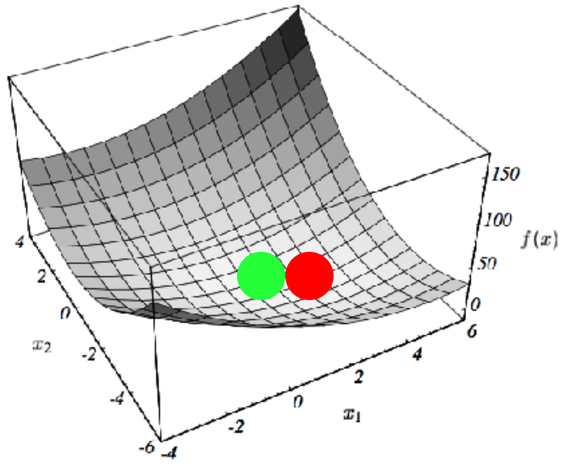


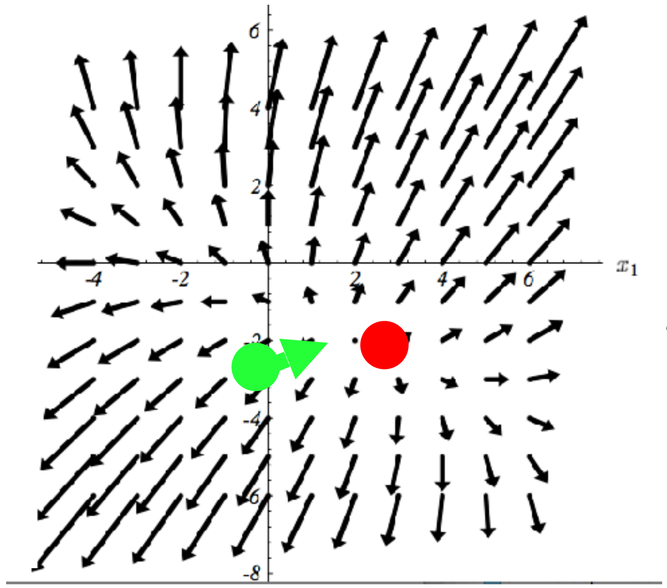


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient Descent Minimization

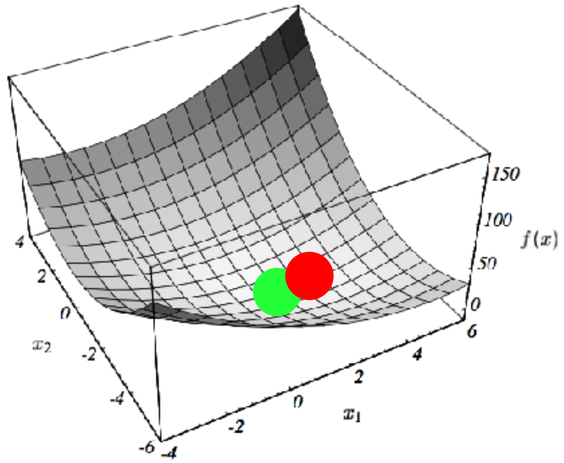


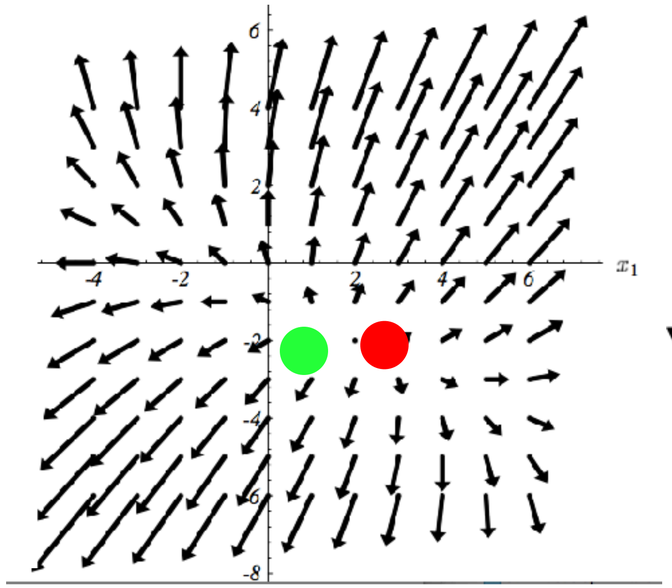


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient Descent Minimization





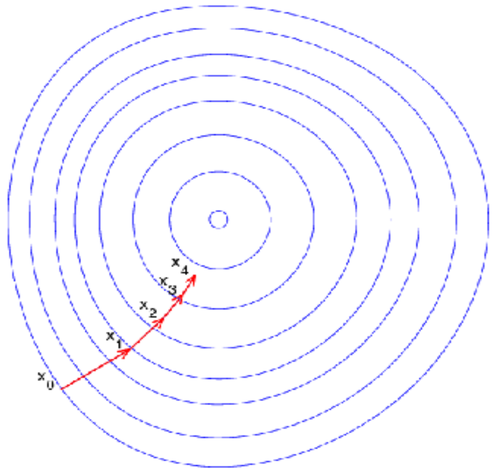
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

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

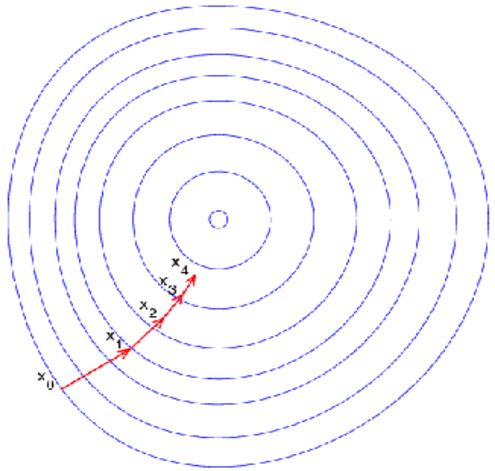


Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

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

We can always make it converge for a convex function.

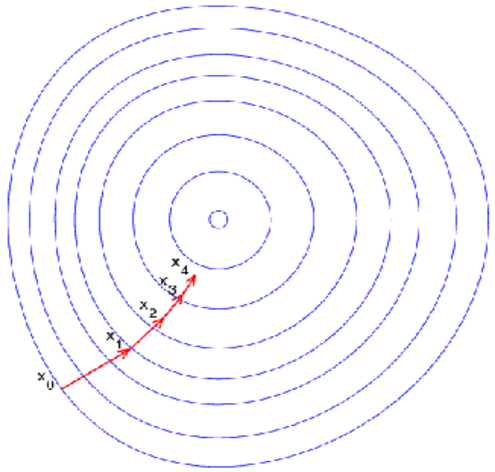


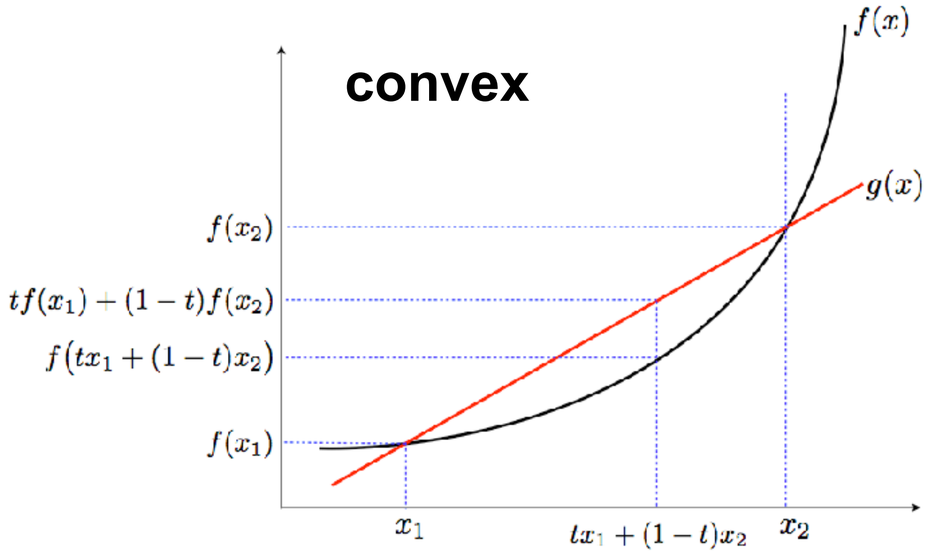
Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

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

We can always make it converge for a convex function.



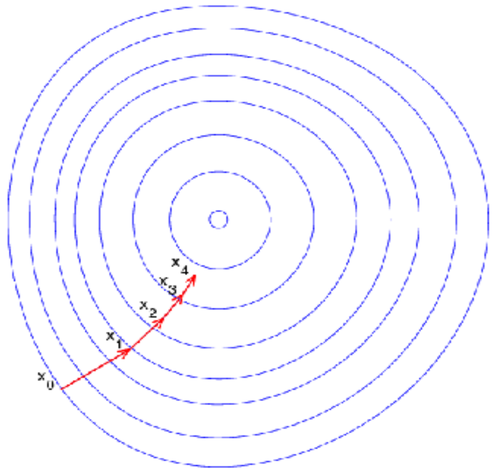


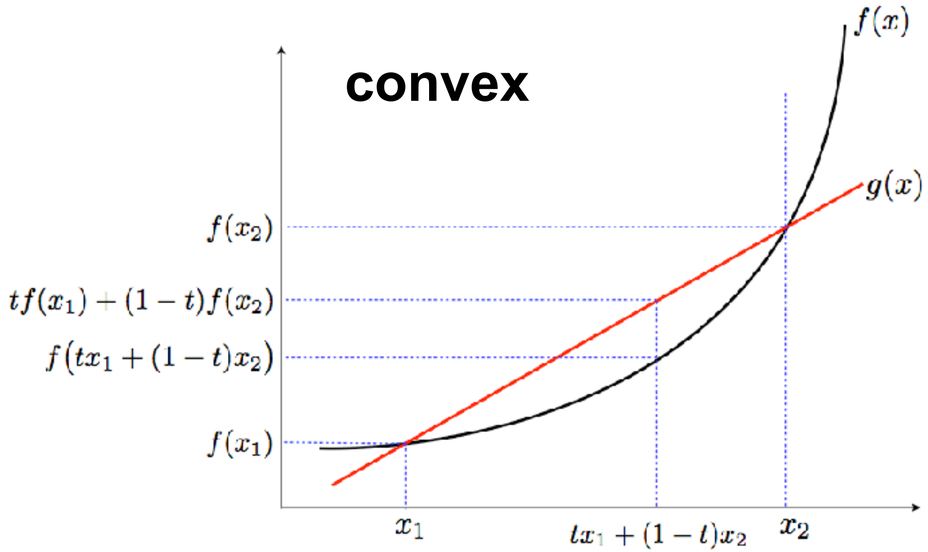
Gradient Descent Minimization

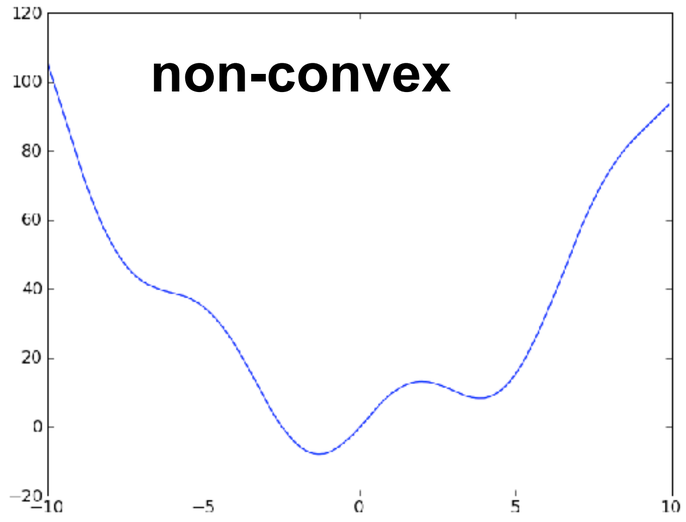
Initialize: $\mathbf{x}_0$

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

We can always make it converge for a convex function.







XOR Problem

XOR Problem

XOR Problem

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

XOR Problem

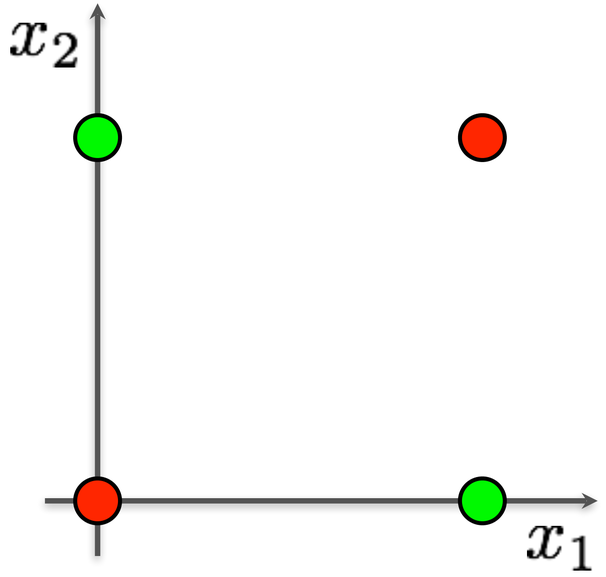
$$y = f(x_1, x_2)$$

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

XOR Problem

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

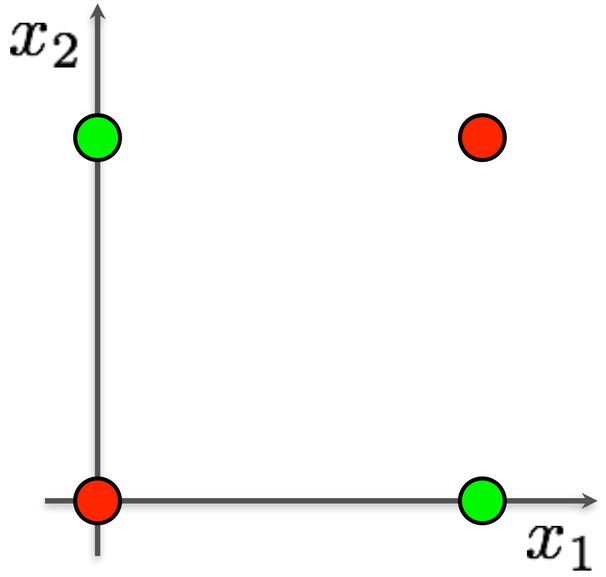
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



XOR Problem

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

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

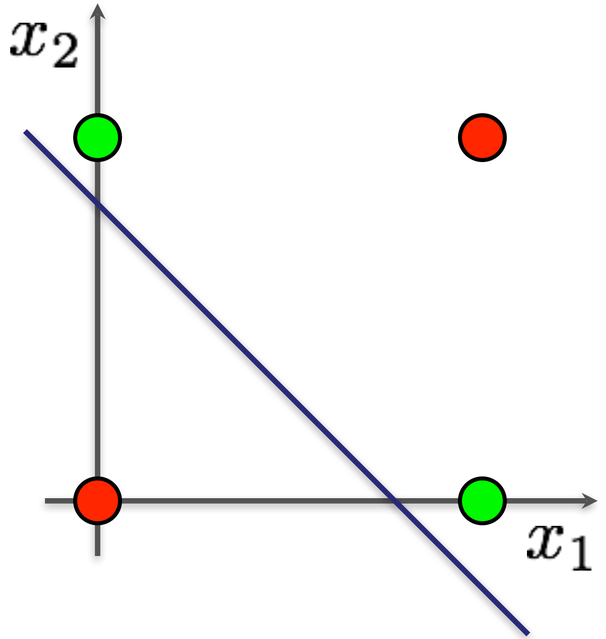


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

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

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

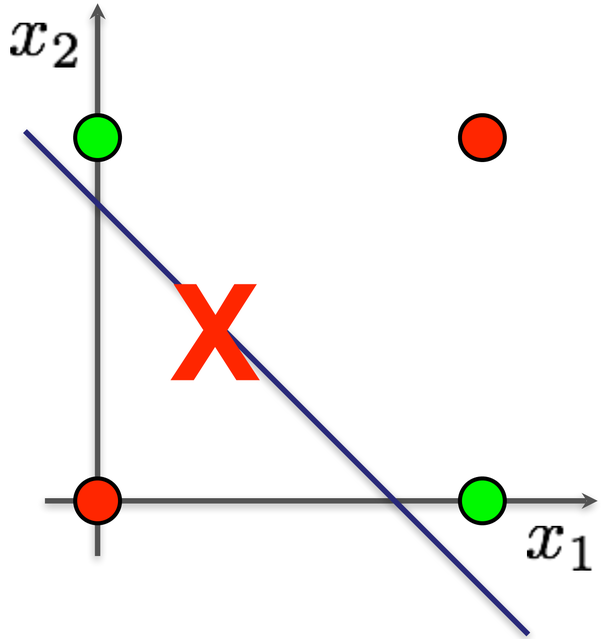


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

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

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

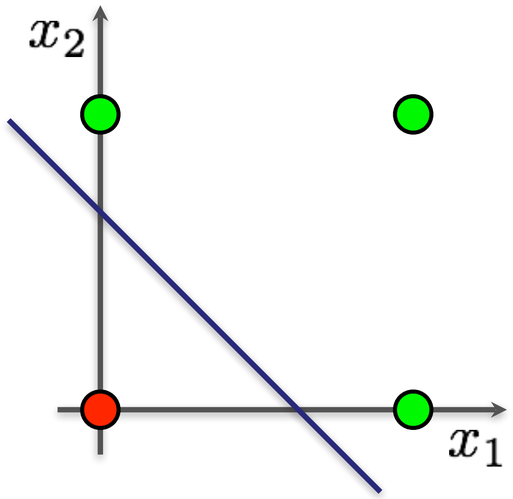


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

XOR Problem

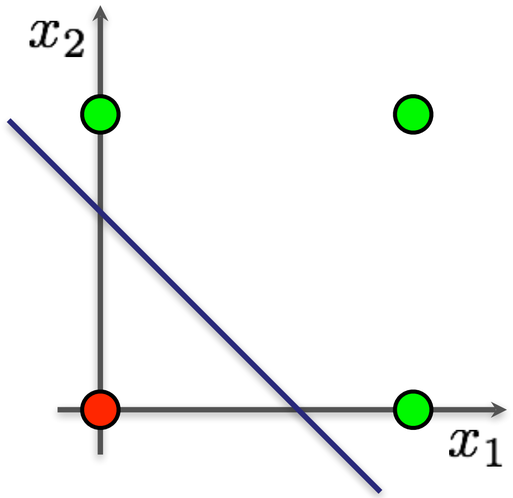
x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1

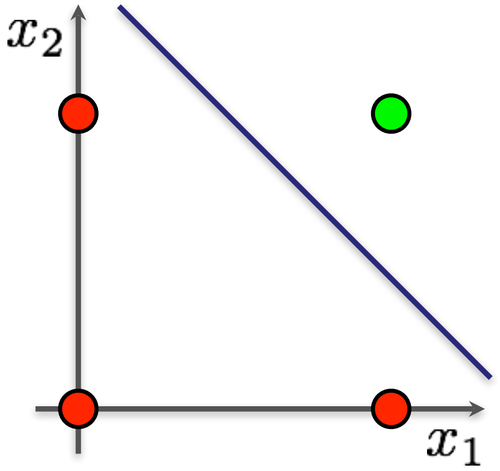


XOR Problem

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

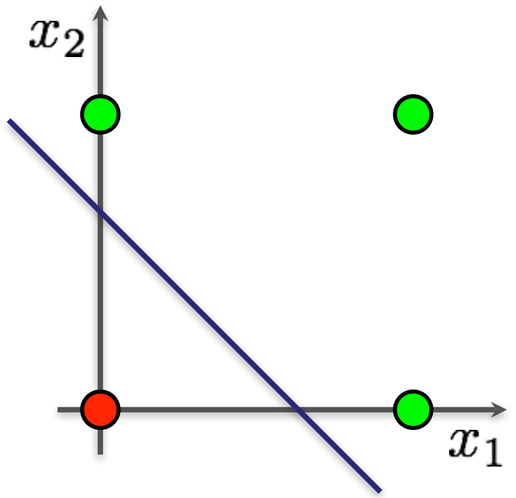
x_1	x_2	z_2
0	0	0
0	1	0
1	0	0
1	1	1



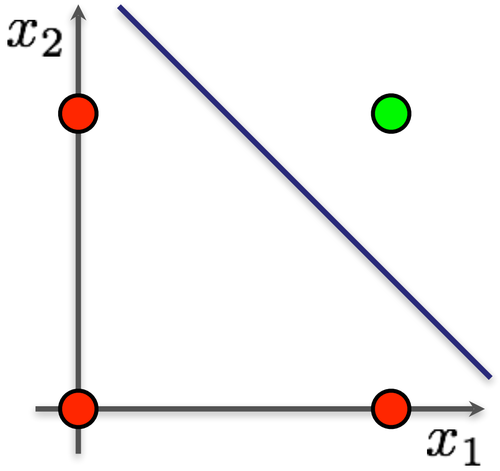


XOR Problem

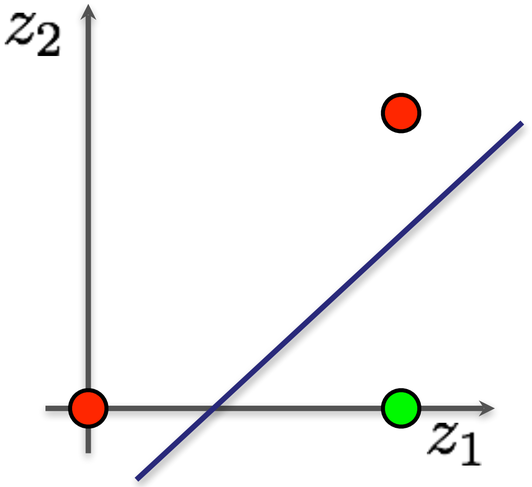
x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1



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



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



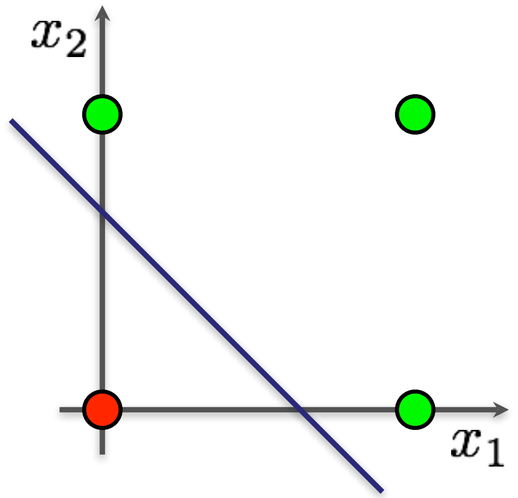
XOR Problem

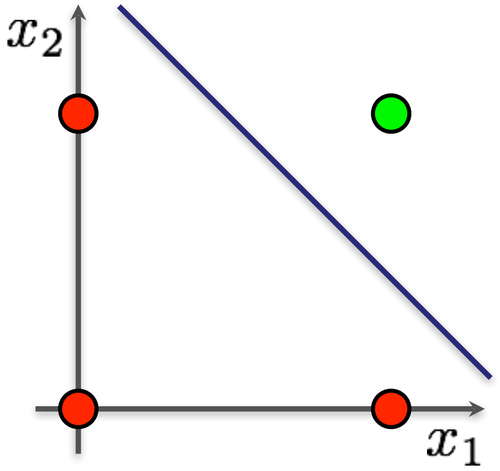
$$y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2))$$

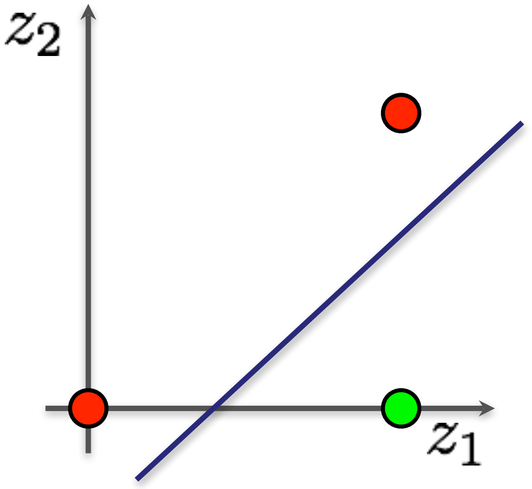
x_1	x_2	z_1
0	0	0
0	1	1
1	0	1
1	1	1

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

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

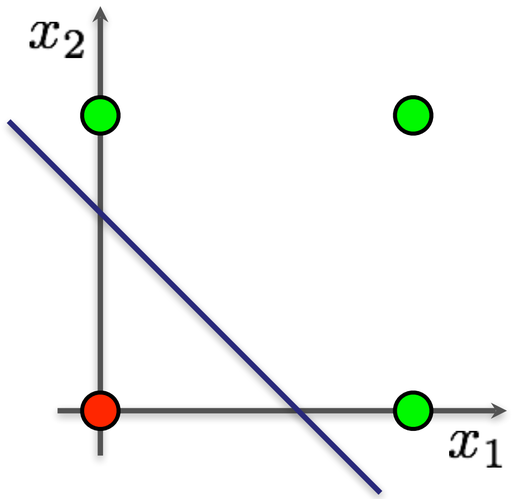


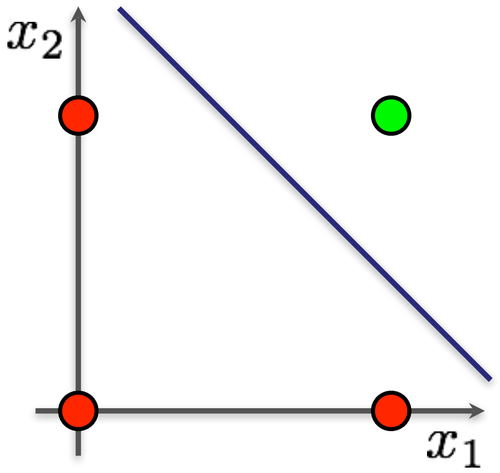


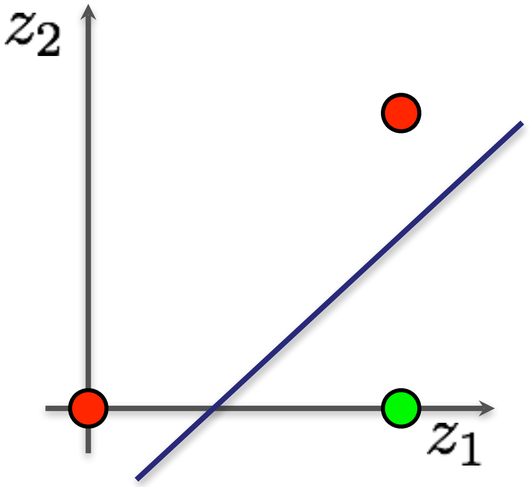


XOR Problem

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

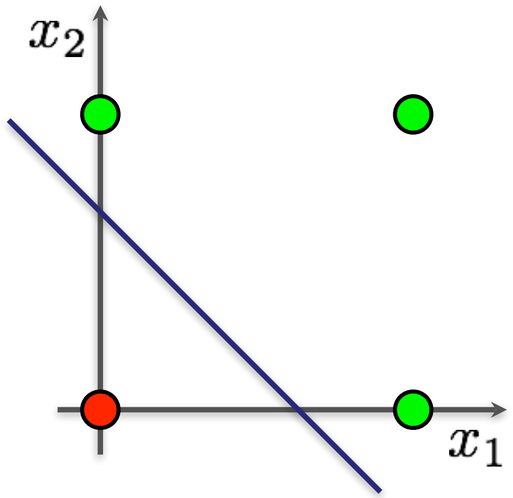


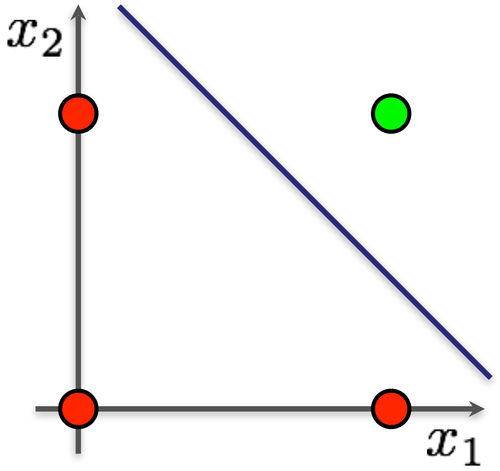


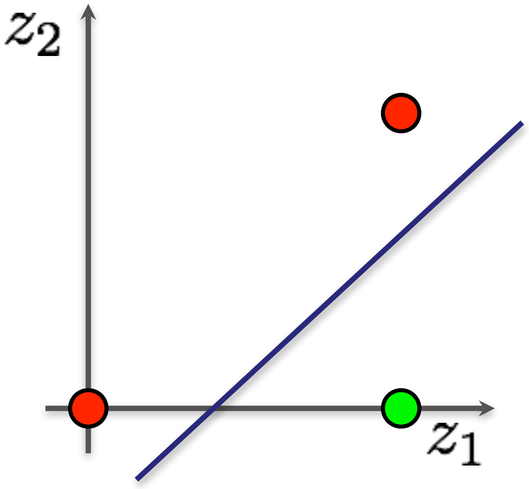


XOR Problem

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

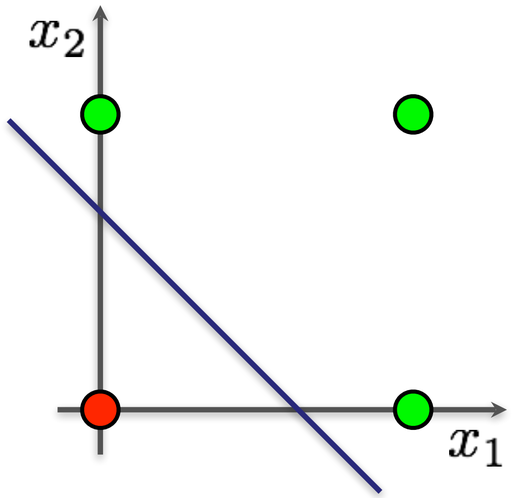


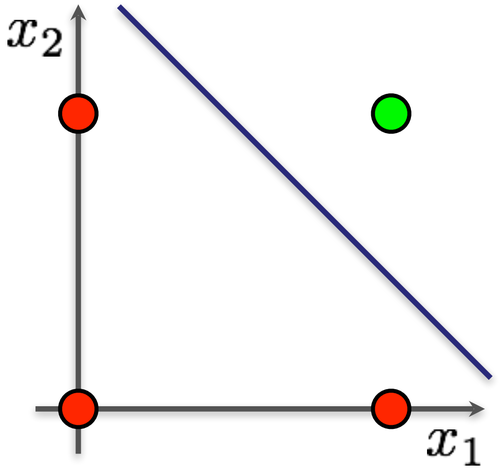


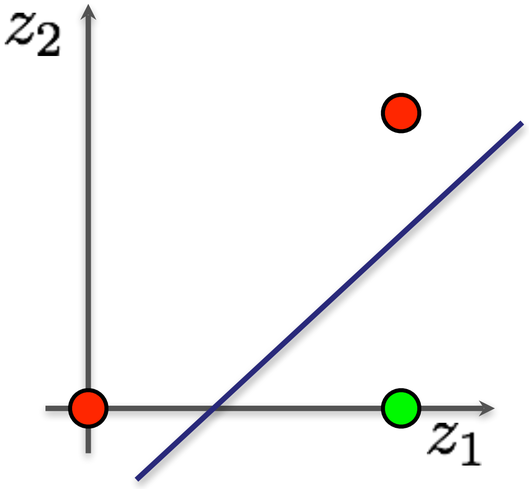


XOR Problem

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

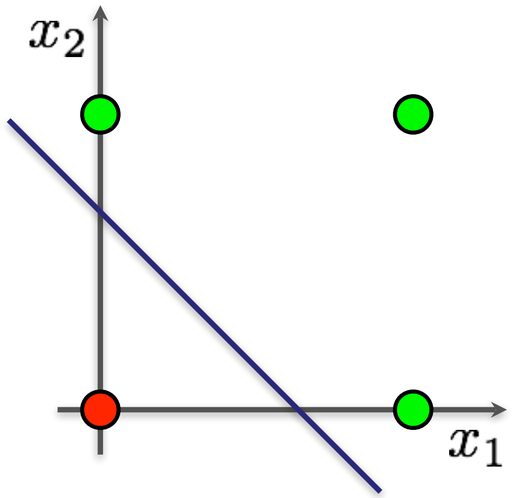


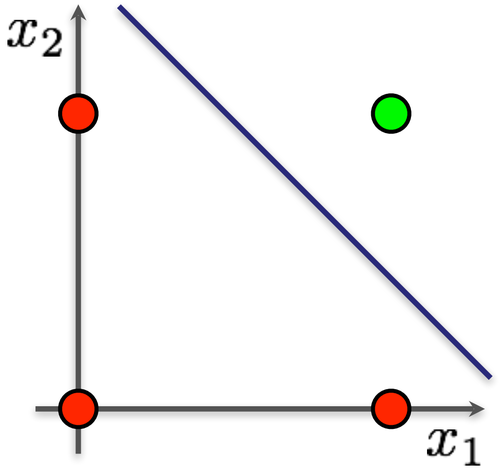


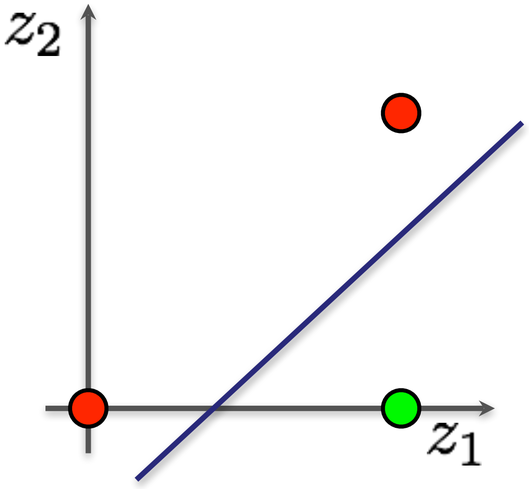


XOR Problem

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

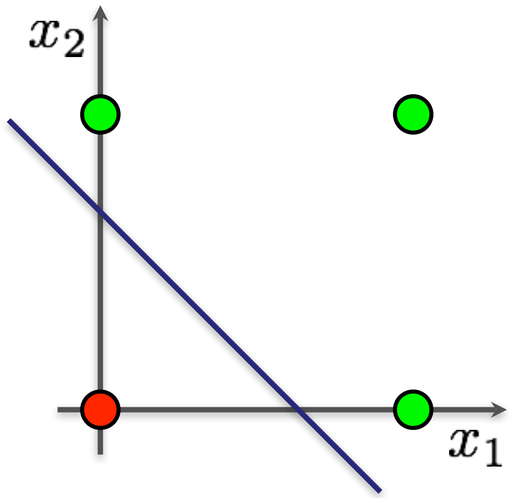


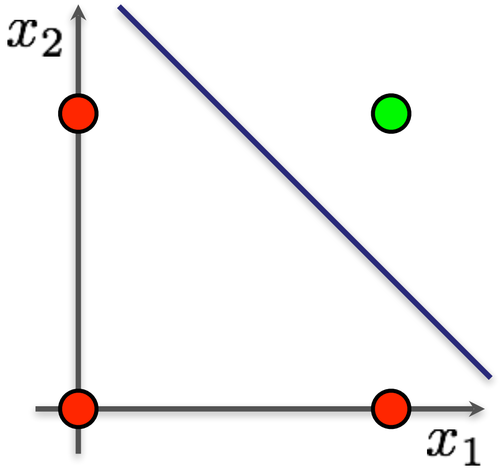


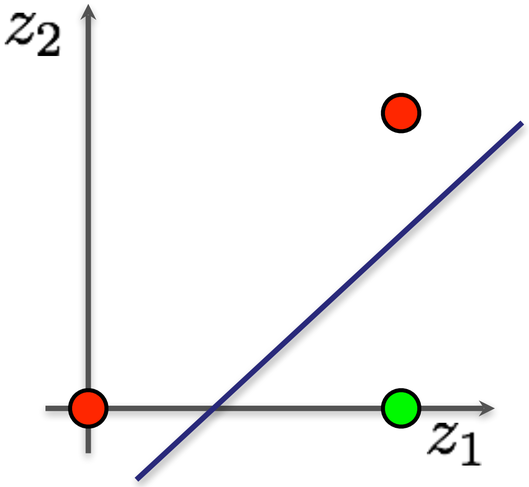


XOR Problem

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





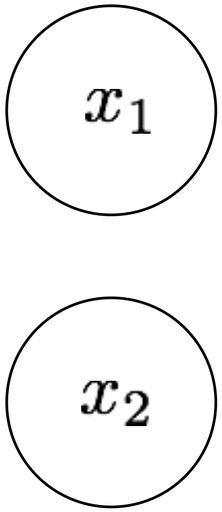


XOR Problem

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

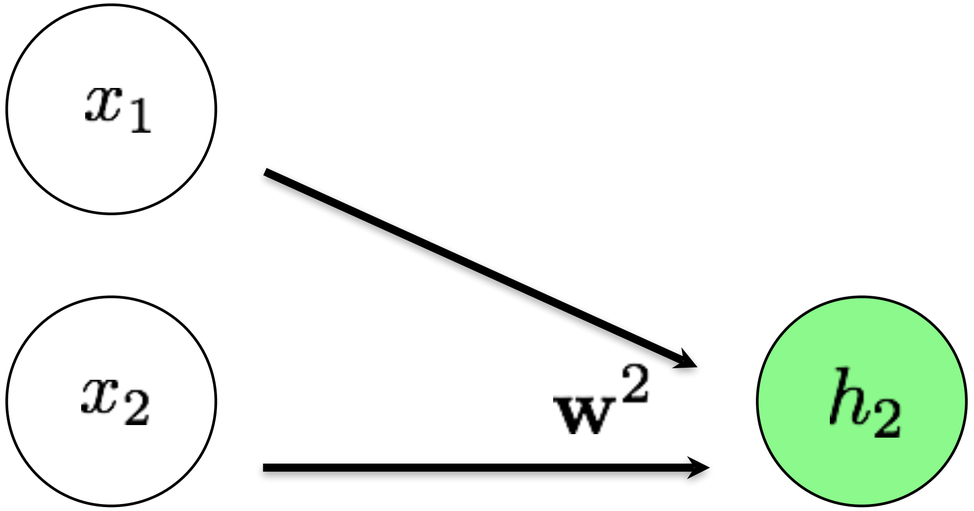
XOR Problem

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



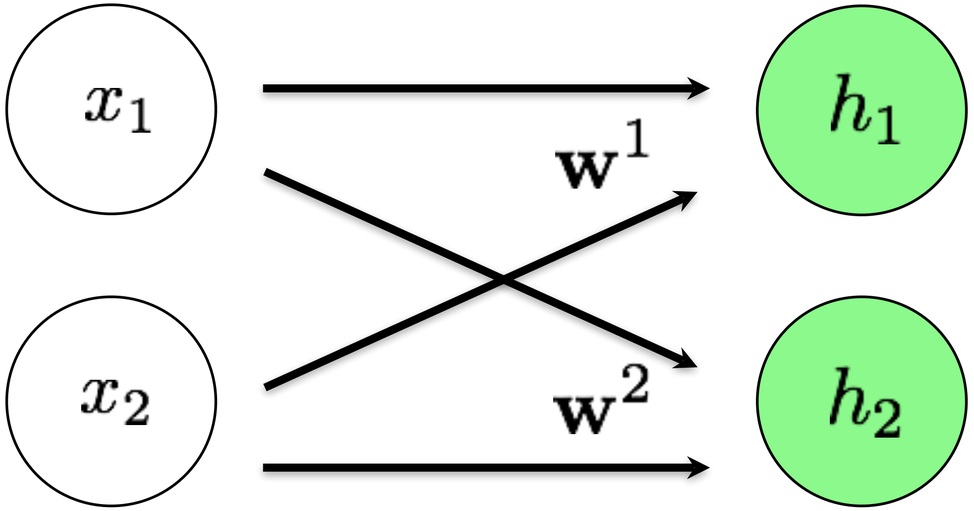
XOR Problem

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



XOR Problem

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



XOR Problem

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