COMP0169: Machine Learning for Visual Computing

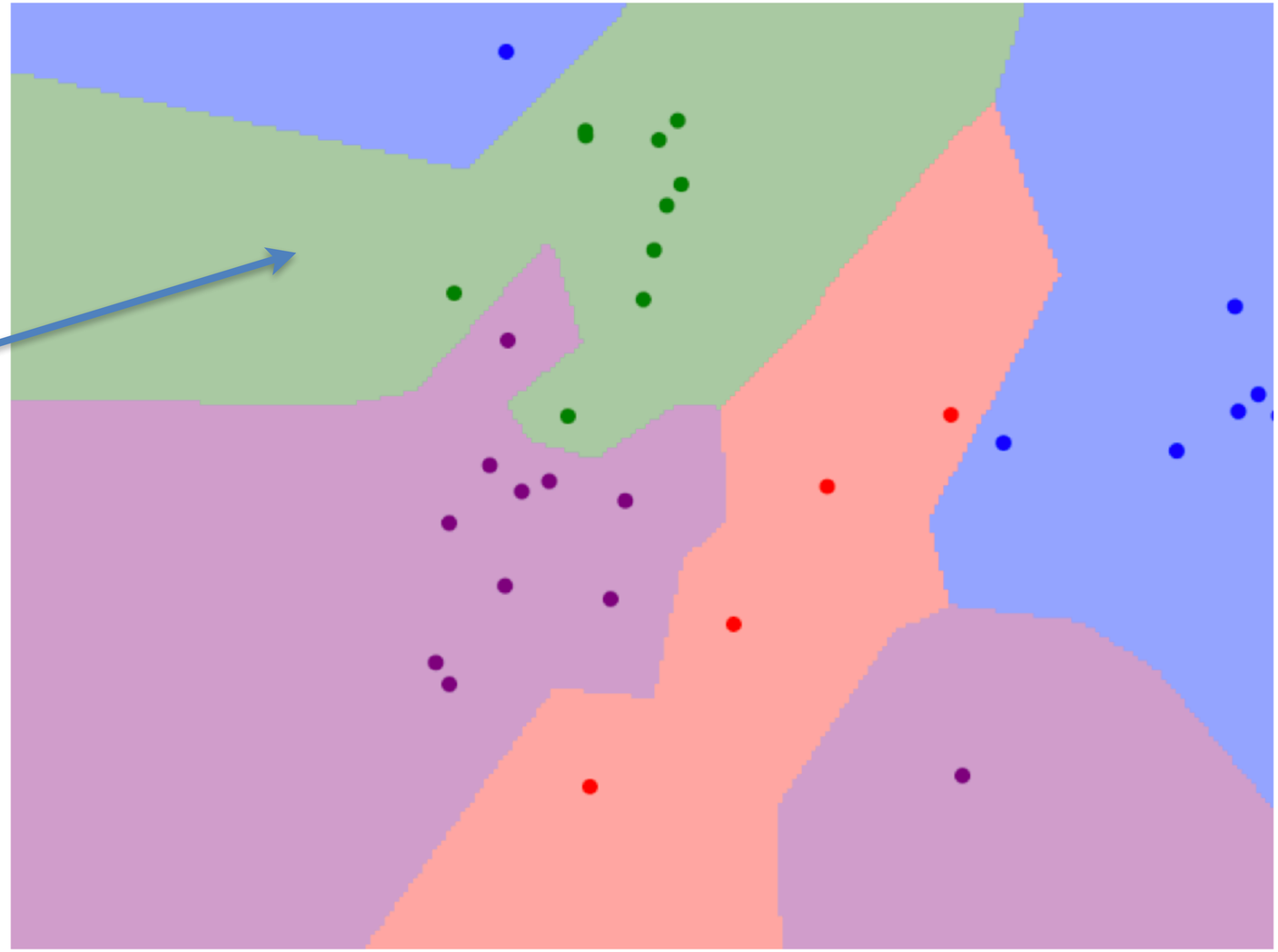
PCA and Linear Classifiers

Lectures will be Recorded

Recap

- Different ML tasks in VC
- Nearest neighbors and kNN classifier
- XOR problem

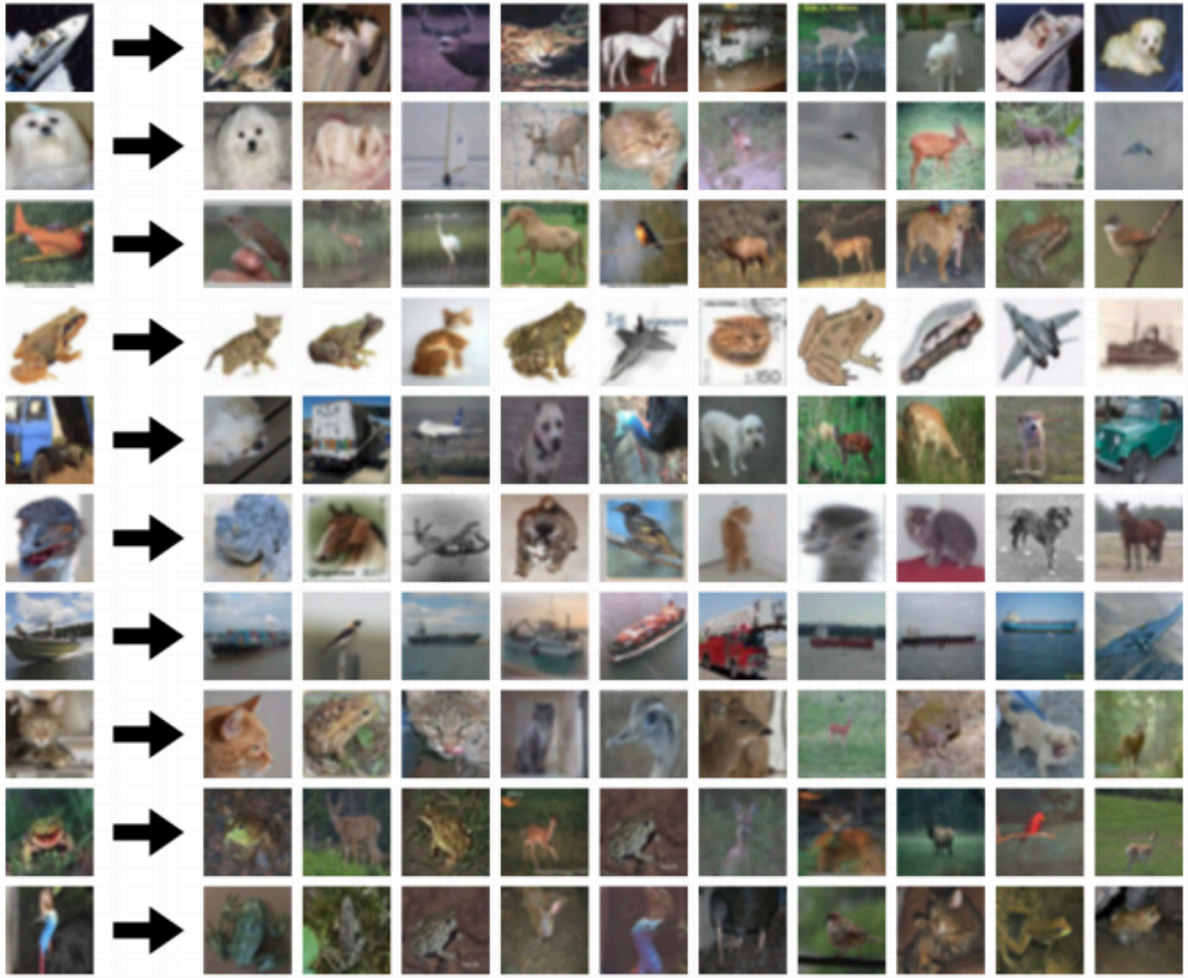
Nearest Neighbor Explained



I_x

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

How Does It Perform?

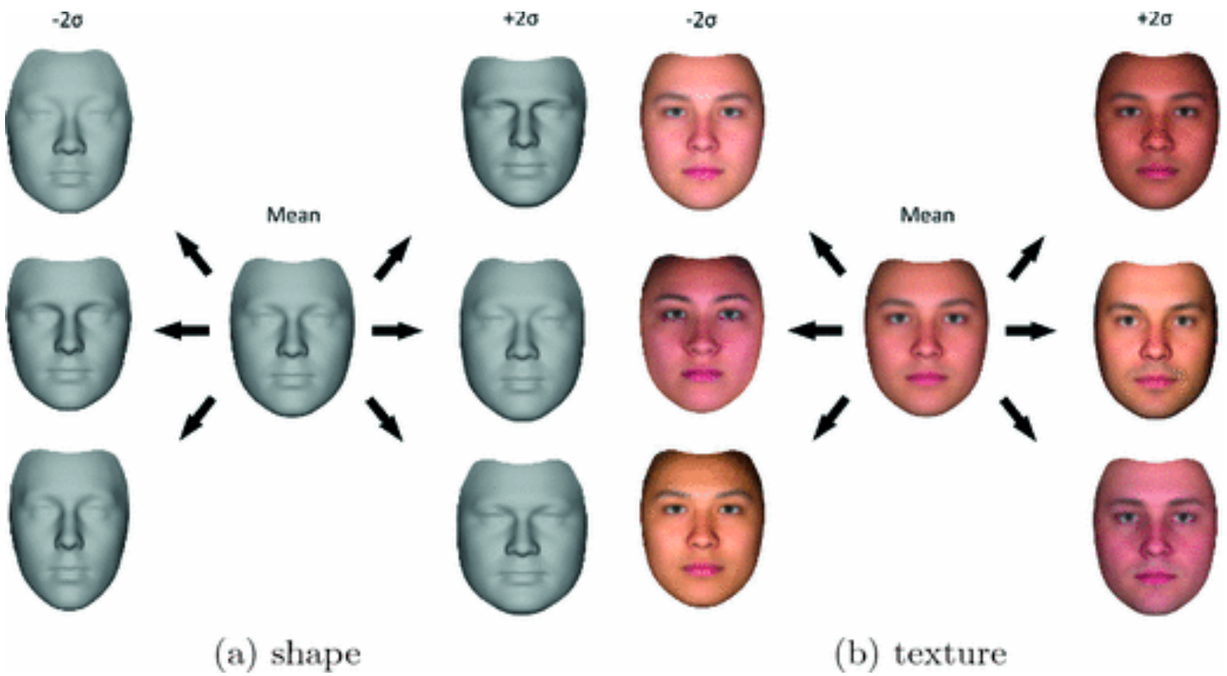


Principal Component Analysis

- **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** $X = U \Sigma W^T$

$$C = X^T X = W \Sigma^2 W^T$$

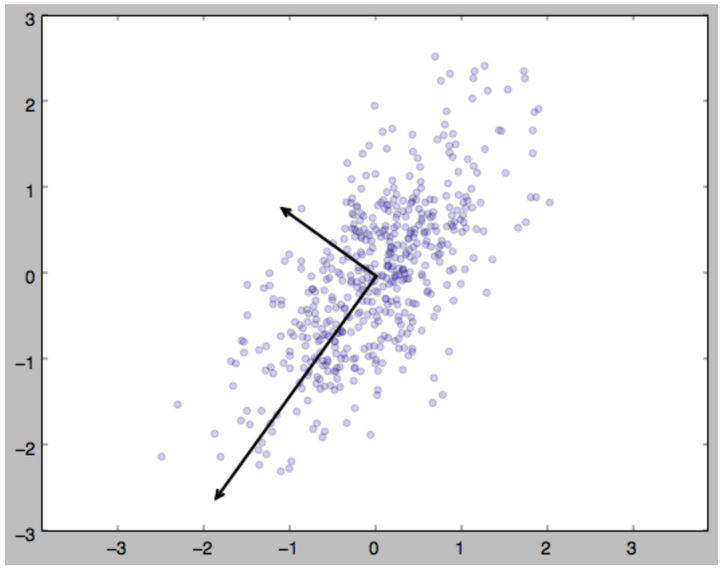
Morphable Faces



Video at http://gravis.dmi.unibas.ch/Sigg99.html

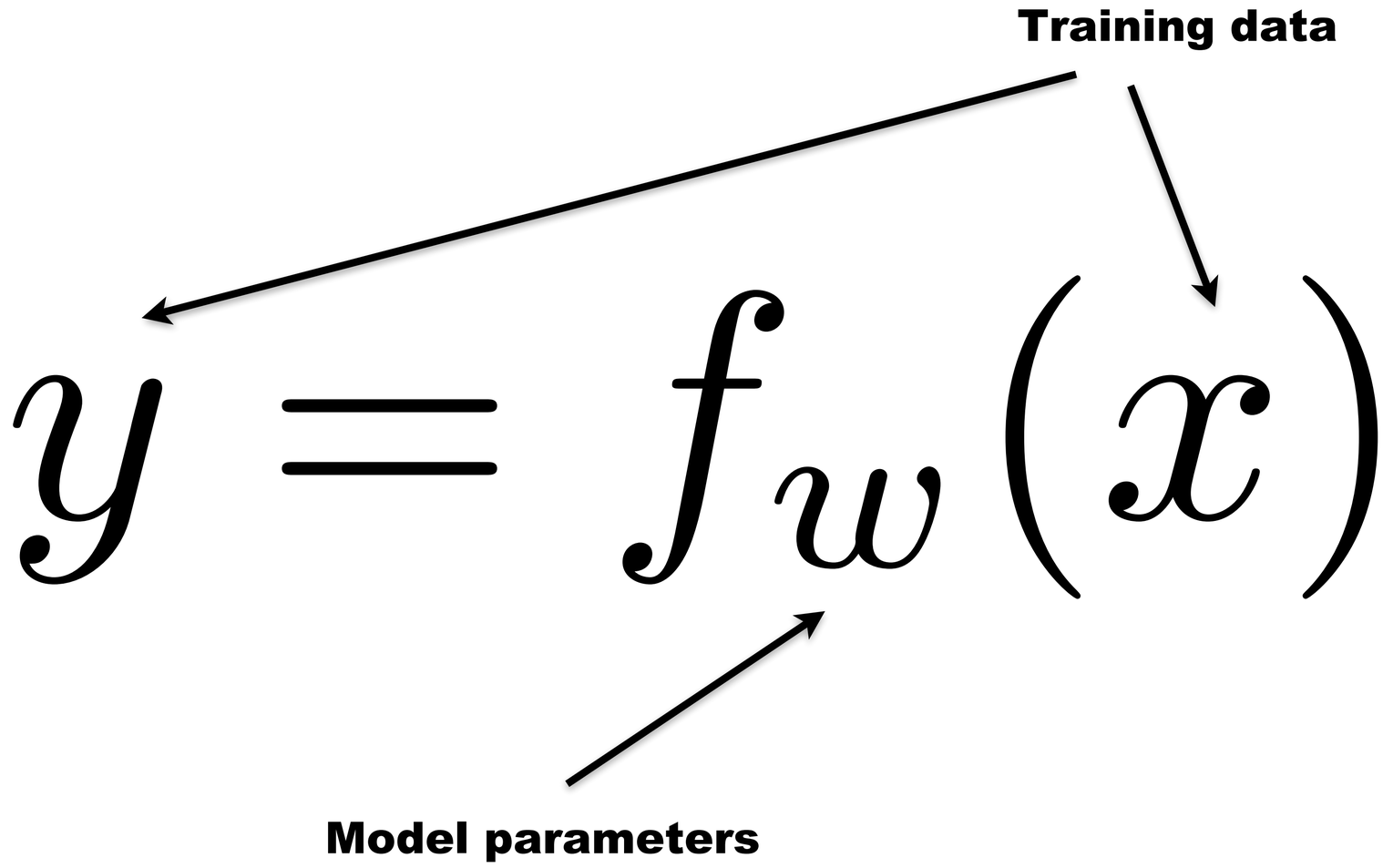
Wood et al., 2021 Arxiv,
Fake it till you make it: face analysis in the wild using synthetic data alone

Code Example

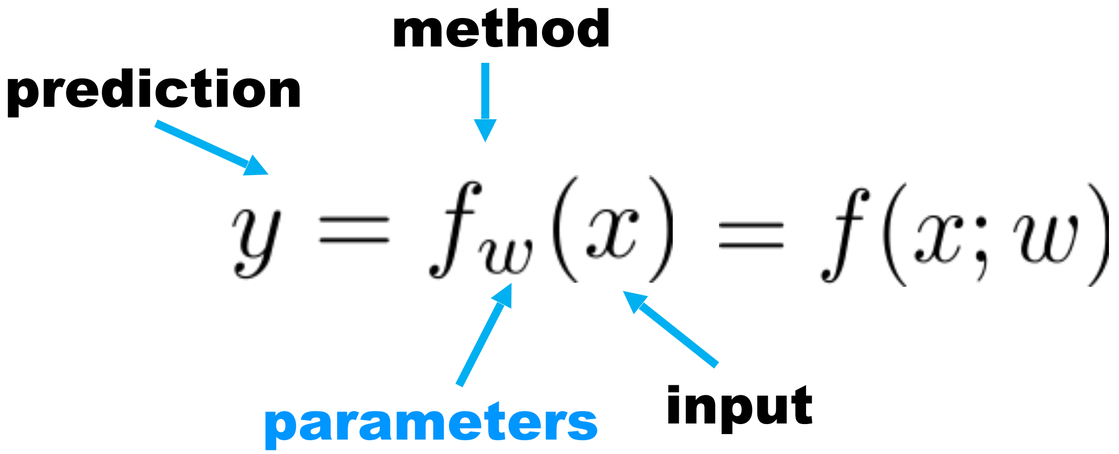


```
mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
matU, sigma, matV = np.linalg.svd(cov_mat)
```

Learning a Function



Learning a Function: Modeling

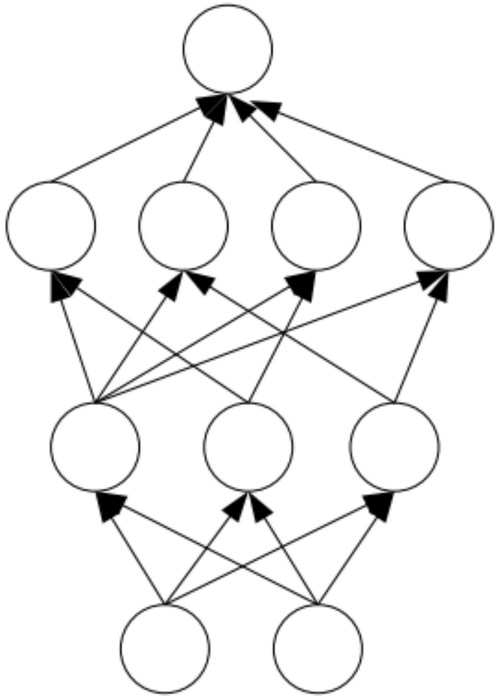


$$w \in \mathbb{R}$$

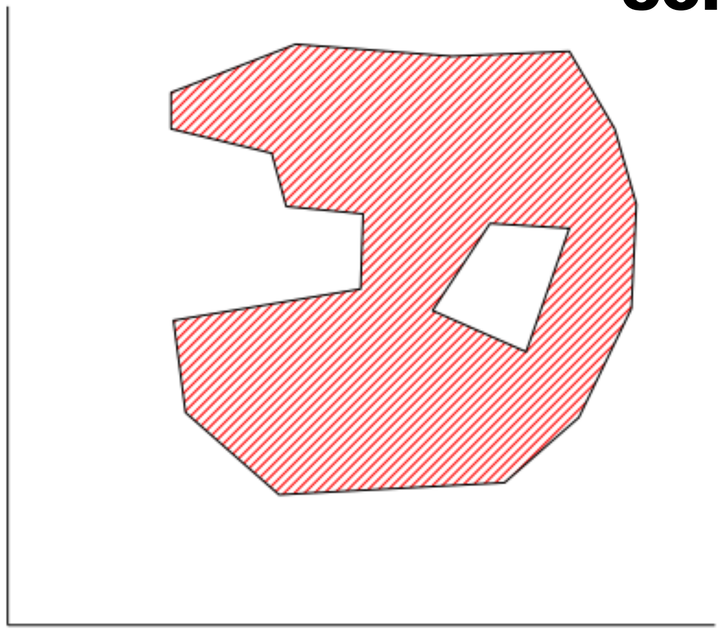
$$\mathbf{w} \in \mathbb{R}^K$$

Combining Simple Functions/Classifiers

3 layers of trainable weights



complex polygons



composition of polygons: convex regions

Regression

1. Least Squares fitting

2. Nonlinear error function and gradient descent

3. Perceptron training

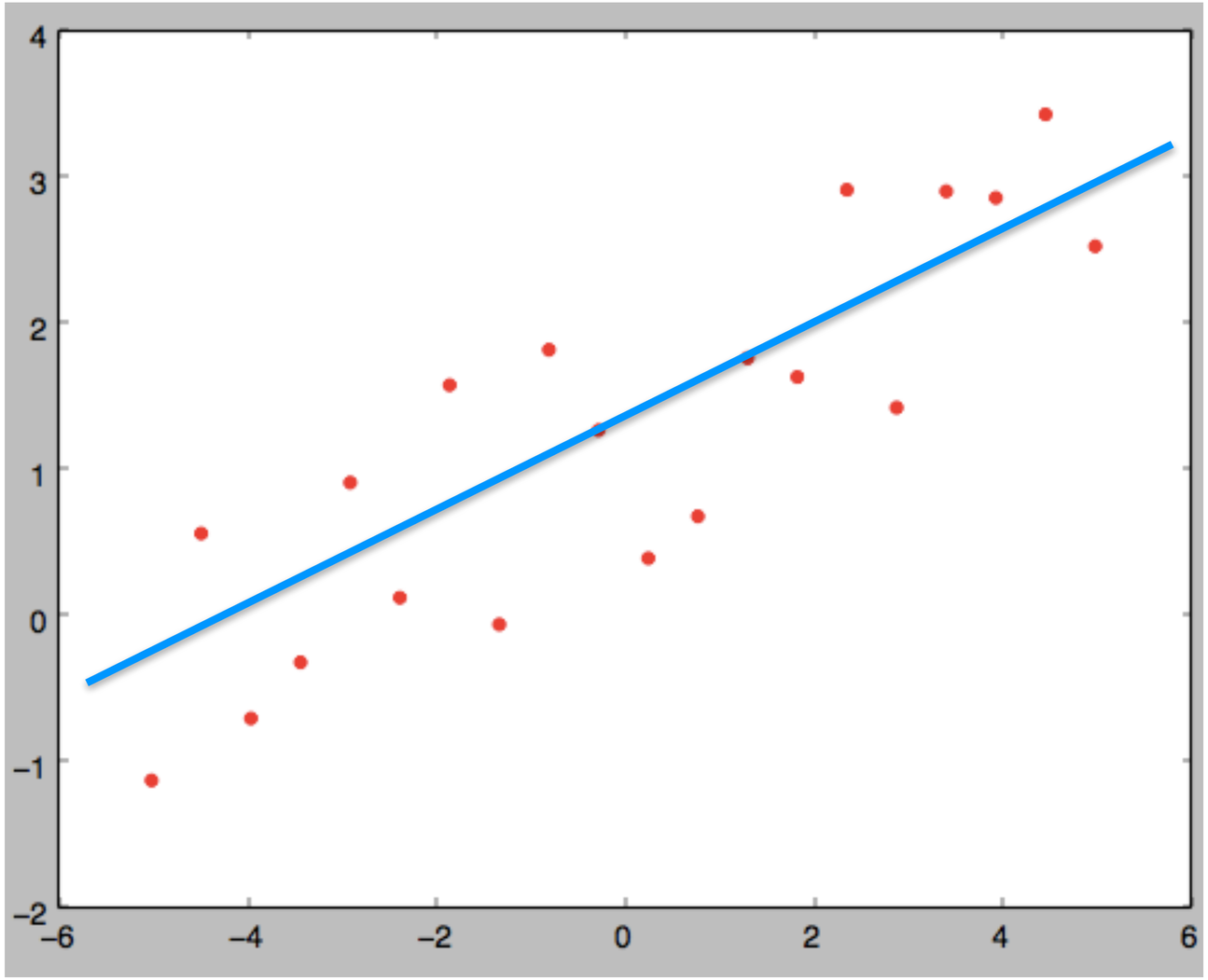
Regression

1. Least Squares fitting

2. Nonlinear error function and gradient descent

3. Perceptron training

Regression: Continuous Output



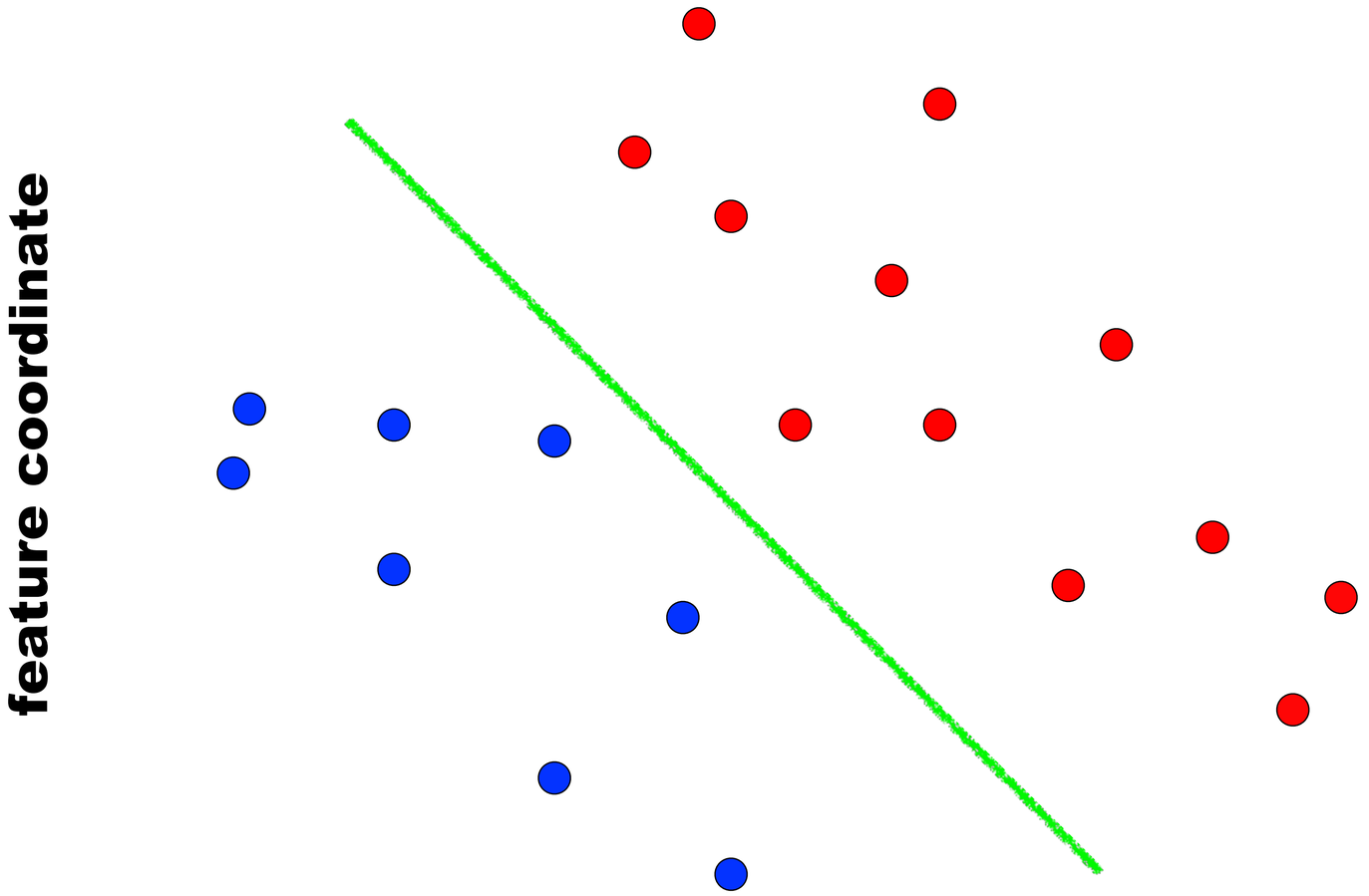
Assumption: Linear Function

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

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

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

Reminder: Linear Classifier



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

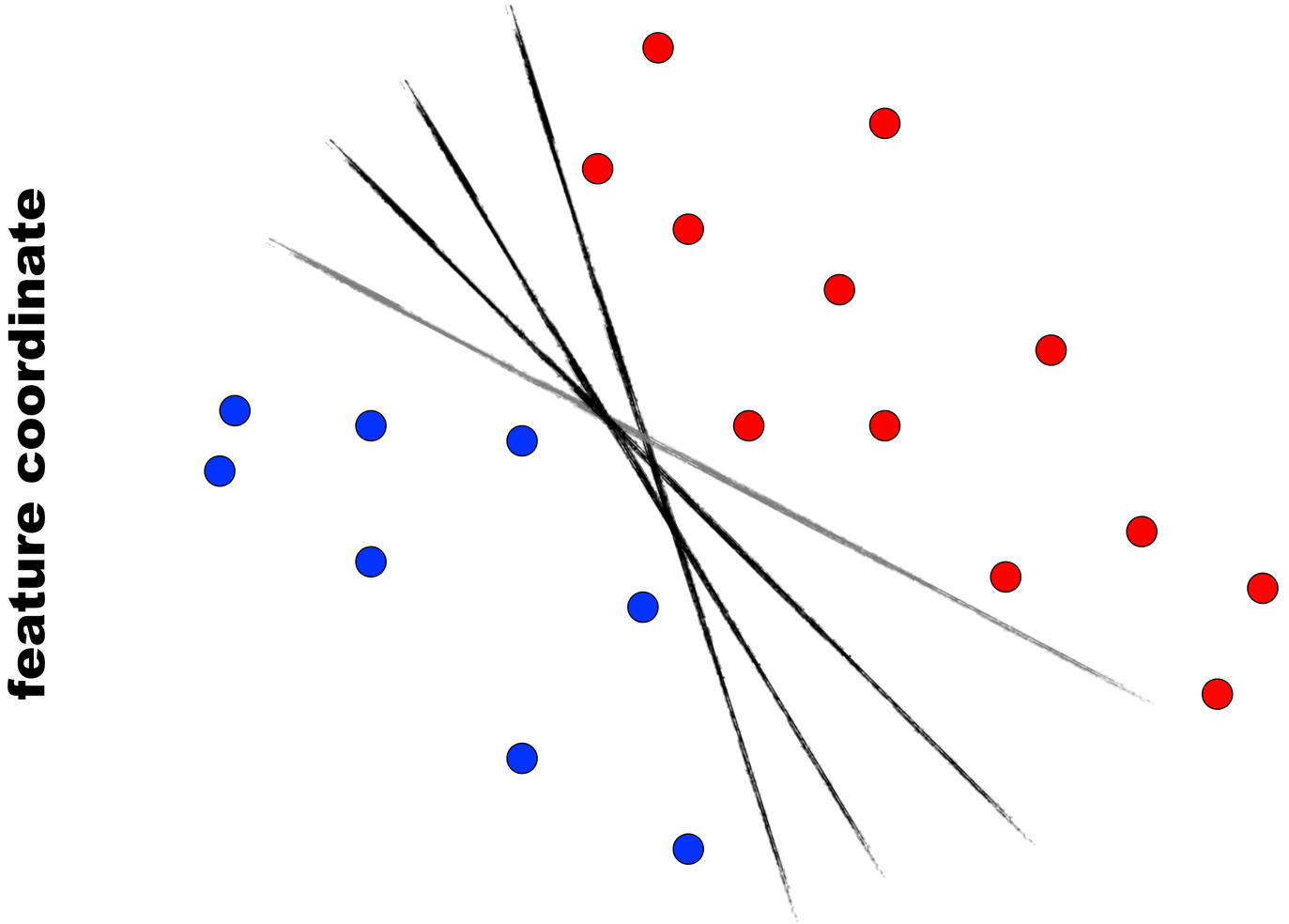
$\mathbf{x}_i$ negative: $\mathbf{x}_i \cdot \mathbf{w} < 0$

supervised setting

labelled input

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

Which Line to Pick?



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

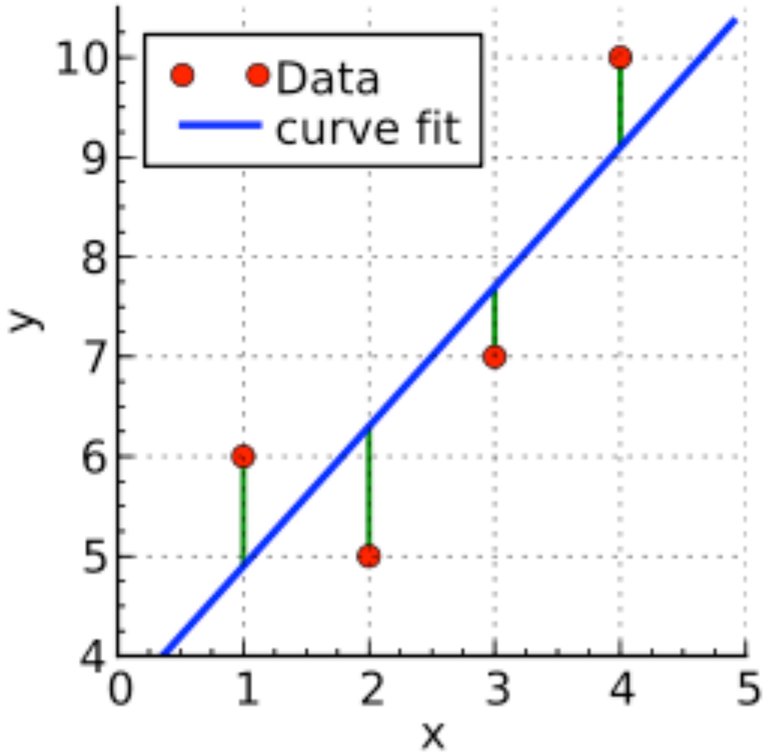
$\mathbf{x}_i$ negative: $\mathbf{x}_i \cdot \mathbf{w} < 0$

supervised setting

labelled input

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

Linear Regression in 1D



Training set:
input–output pairs

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

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

Linear regression in 1D

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

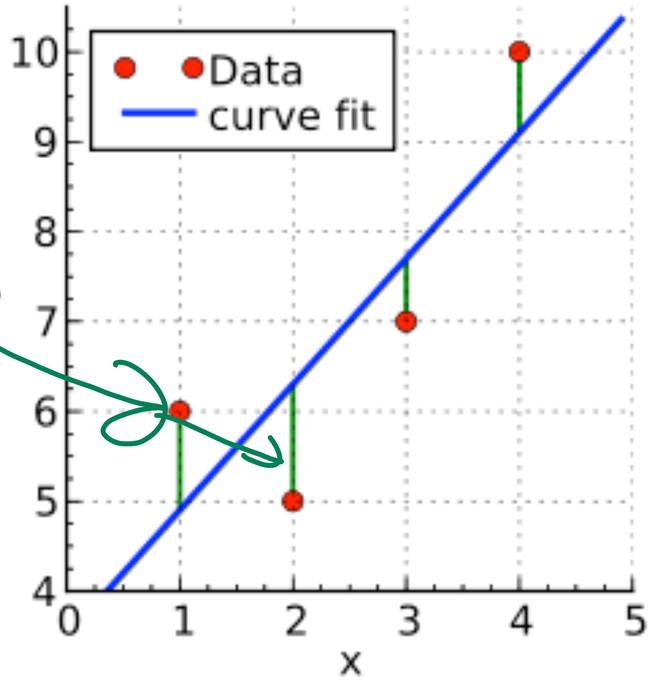
w_0 bias

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

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

noise

green vector



Sum of Square Errors (MSE without the mean)

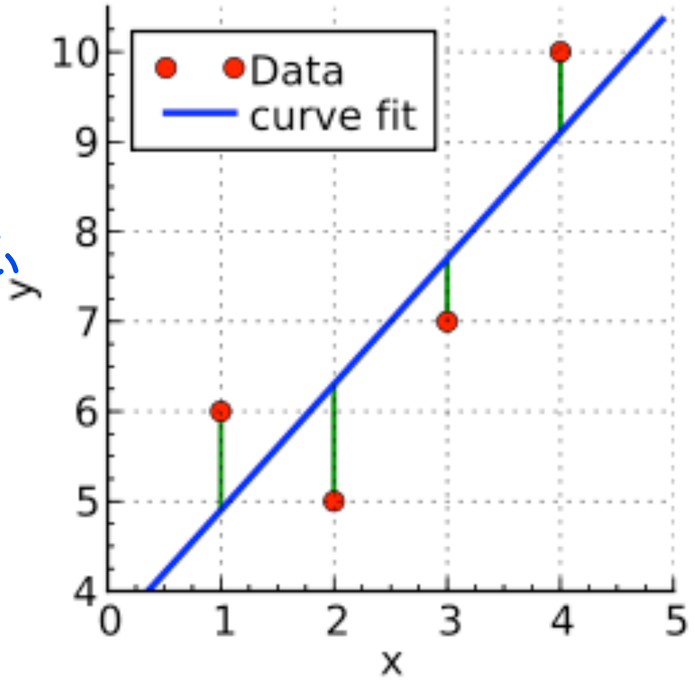
$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

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

In two variables:

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



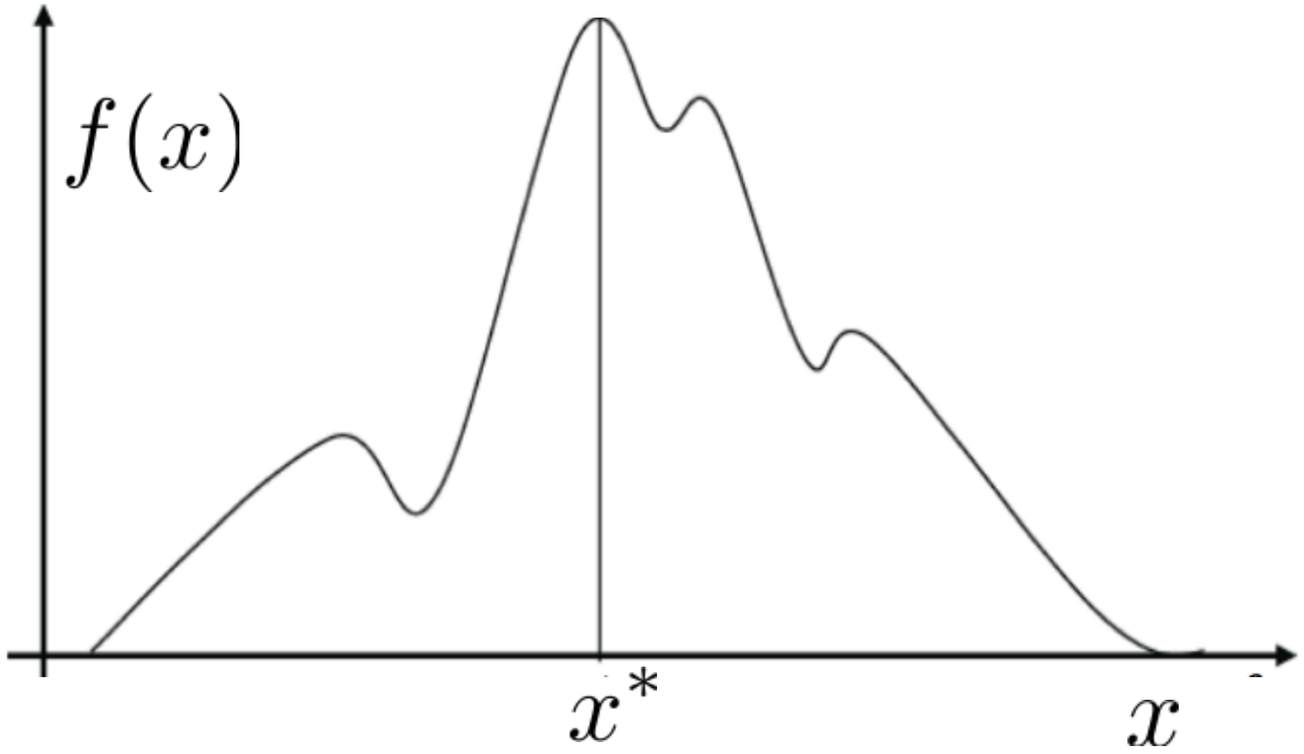
Question: what is the best (or least bad) value of w?

Line Fitting

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

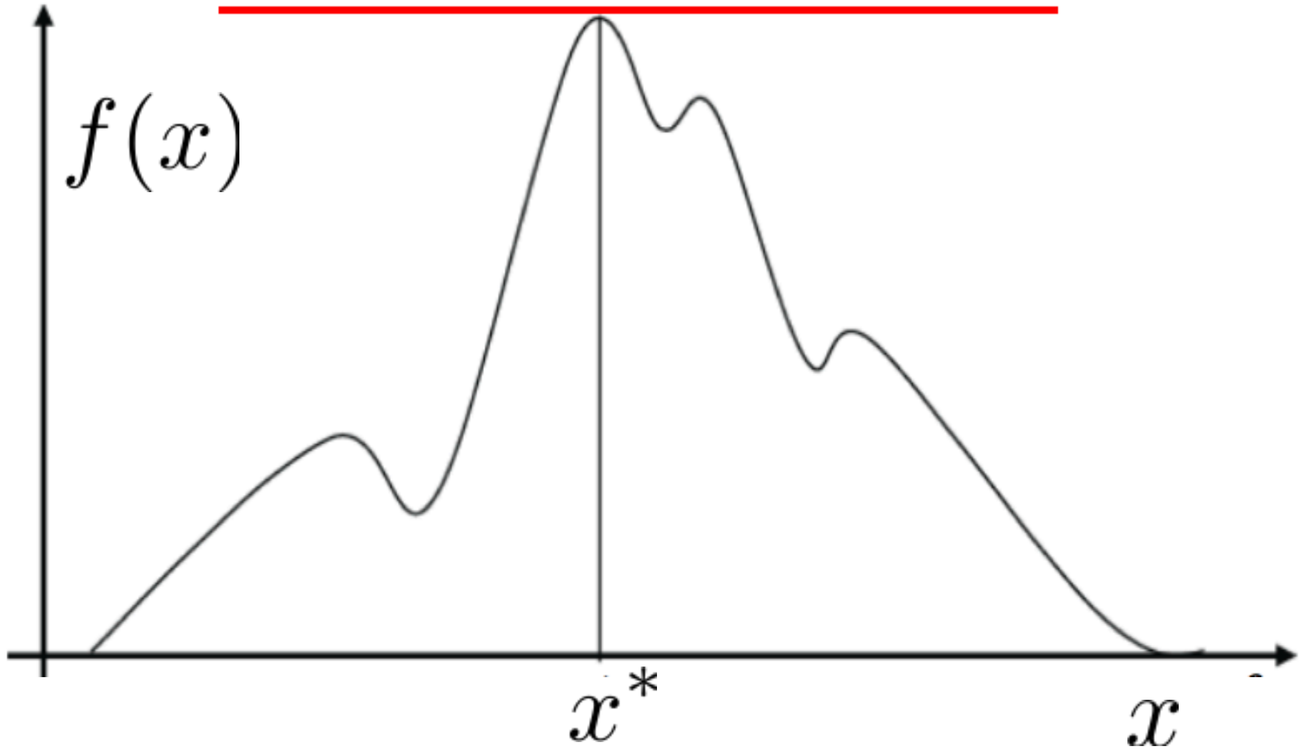
$$\mathbf{X} = \begin{bmatrix} x_0^1 & x_1^1 \\ \vdots & \vdots \\ x_0^N & x_1^N \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix}$$

Calculus 101



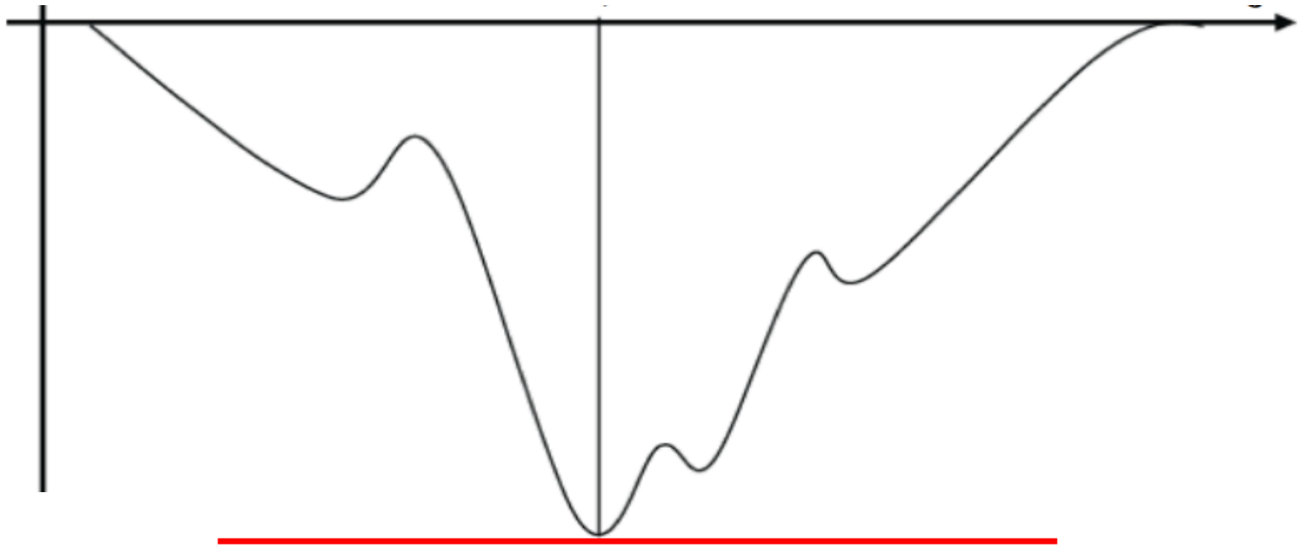
$$x^* = \text{argmax}_x f(x)$$

Local Extrema Condition



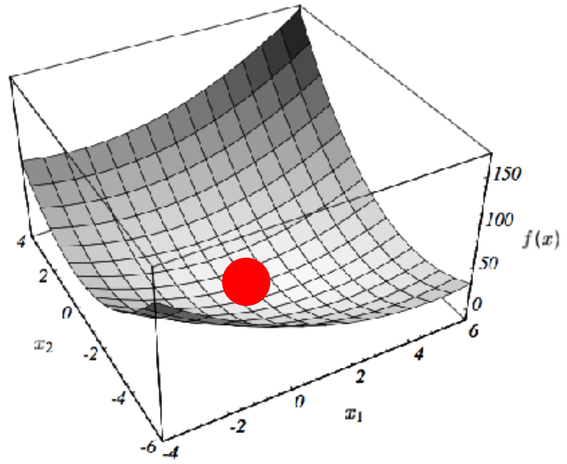
$$x^* = \text{argmax}_x f(x) \quad \rightarrow \quad f'(x^*) = 0$$

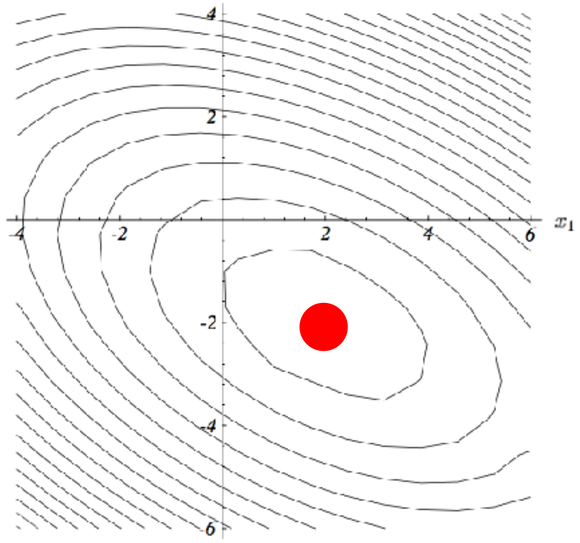
Local Extrema Condition

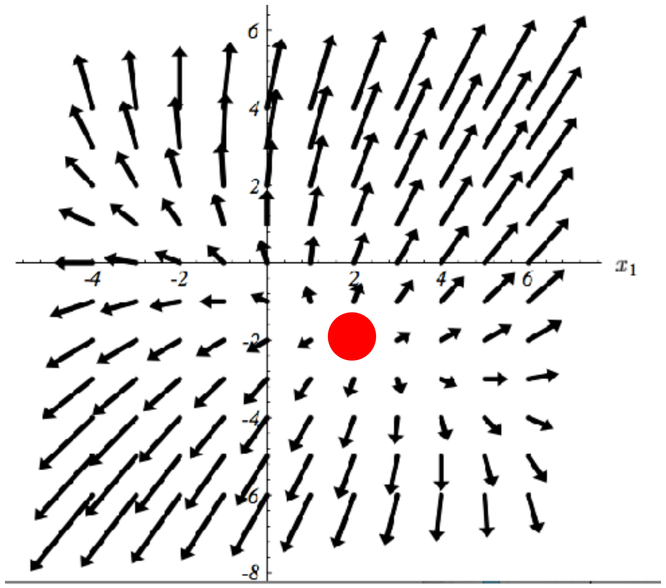


$$x^* = \text{argmax}_x f(x) \quad \rightarrow \quad f'(x^*) = 0$$

Vector Calculus 101







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

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

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

2D function graph **isocontours** **gradient field**

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

Line Fitting

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

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

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

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

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

Line Fitting (continued)

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

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

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

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

Line Fitting (continued)

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

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

2x2 system of equations

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

Line Fitting (continued)

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

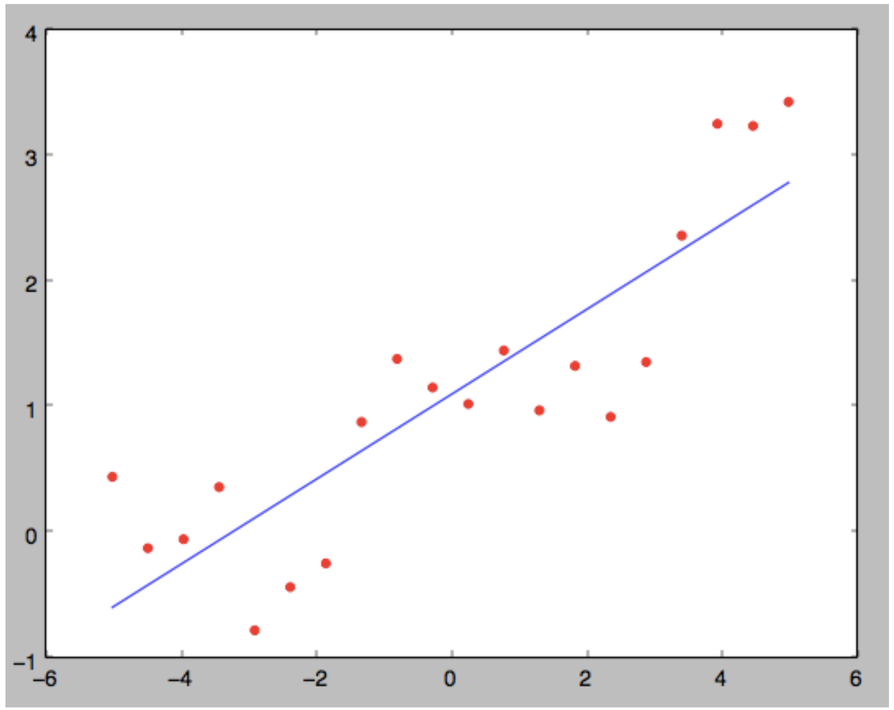
$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w}$$

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

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

Normal Equation

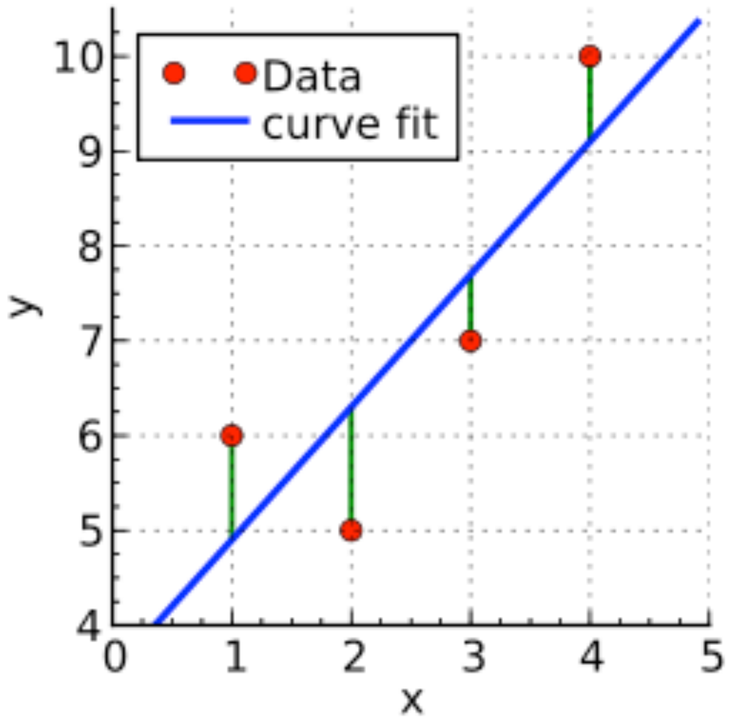
Code Example

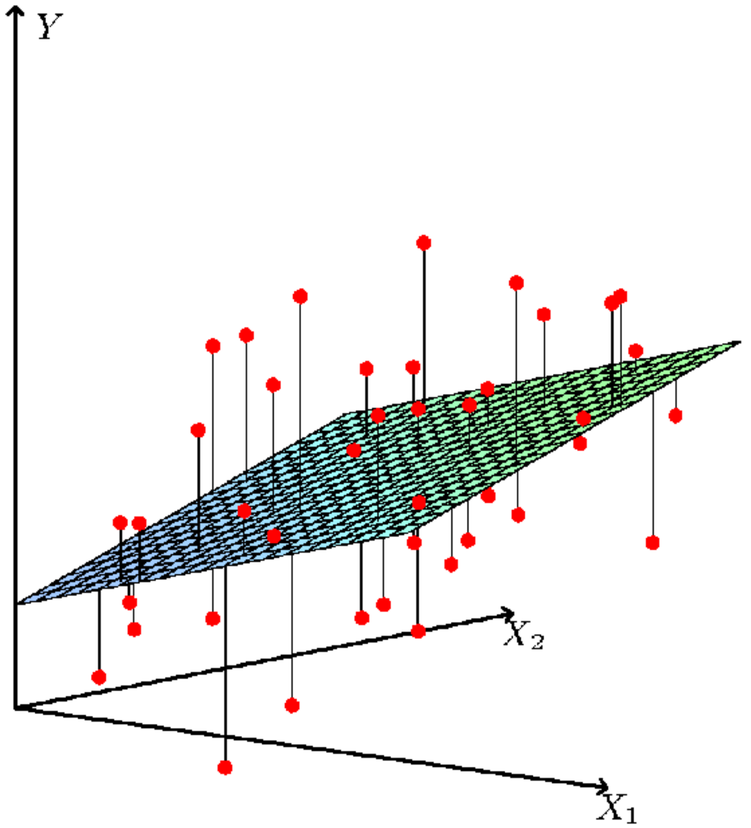



```python
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

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

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

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

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


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

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

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

Linear Regression (Line/Plane Fitting)





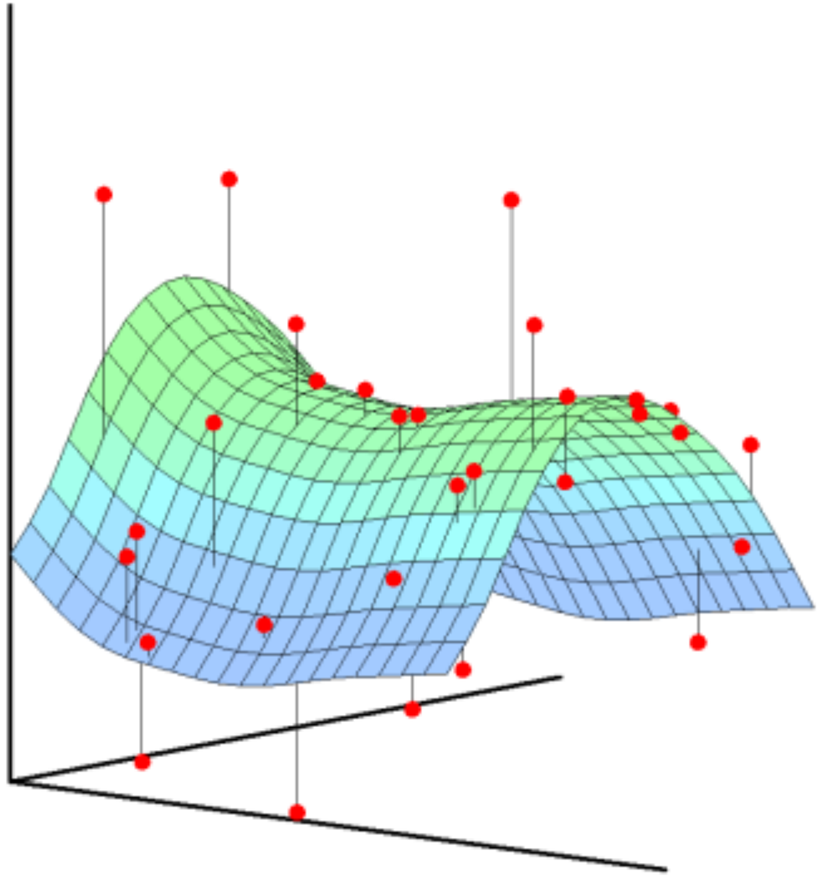
LS Solution for Regression

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

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

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

Generalized Linear Regression

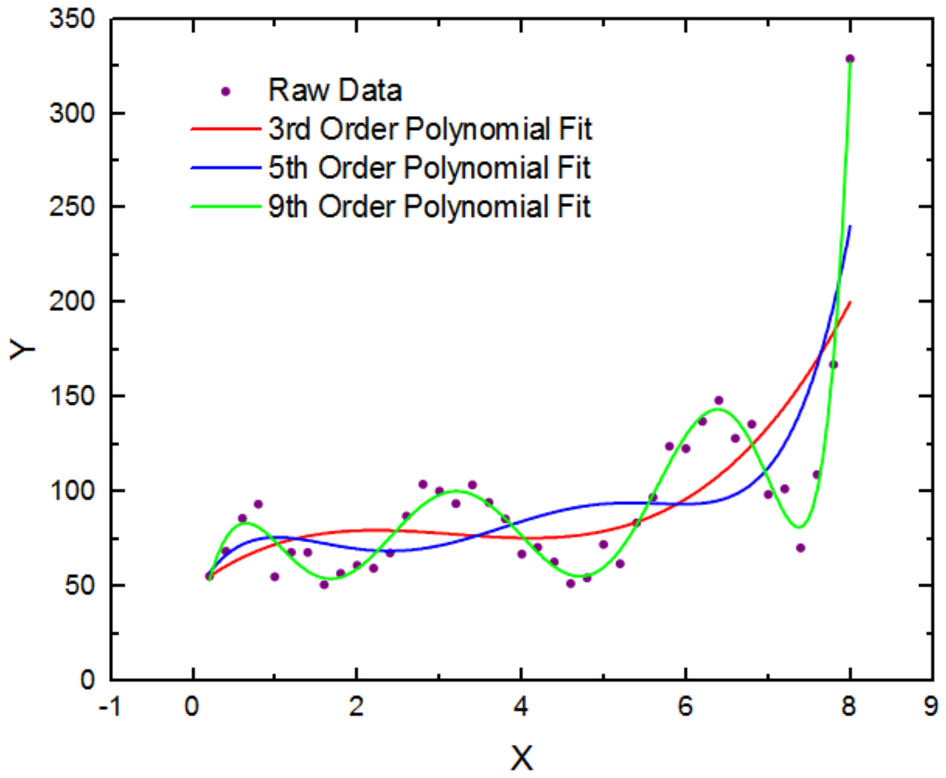


$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}$$

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

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

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

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

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

LS Solution for Linear Regression

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

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

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

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

LS Solution for Generalized Linear Regression

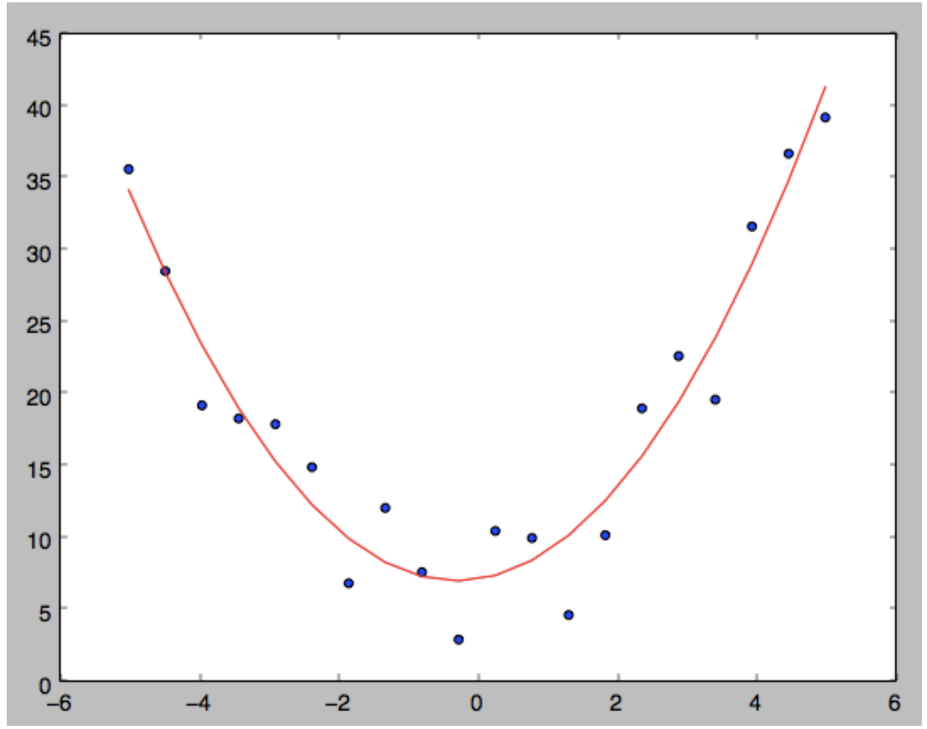
$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} + \boldsymbol{\epsilon}$$

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

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

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

Code Example



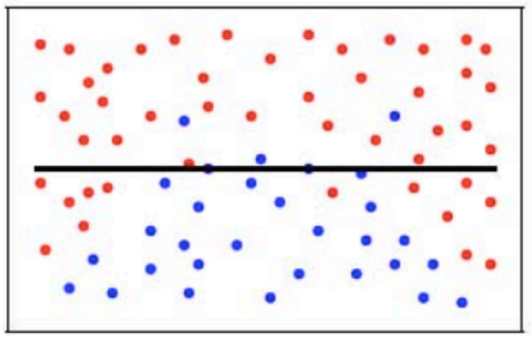

```
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# 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()
```

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

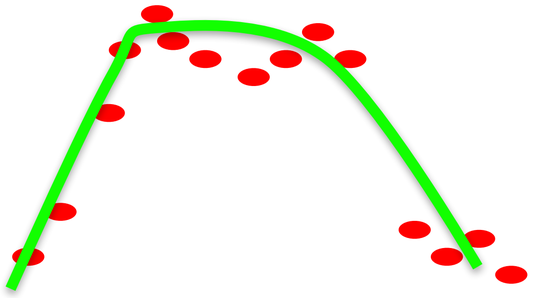
Underfitting vs. Overfitting

Underfitting

classification



regression



Regularized Linear Regression

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

residual vector

这里Φ是X构成的矩阵

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

linear regression: minimize model error

Complexity term: (regularizer)

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

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

"data fidelity" **complexity**

minimum remains to be determined

scalar, remains to be determined

Least Squares Solution

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

Condition for minimum:

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

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

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

Ridge regression: L2-regularized Linear Regression

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

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

as before, for linear regression

identity matrix

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

Condition for minimum:

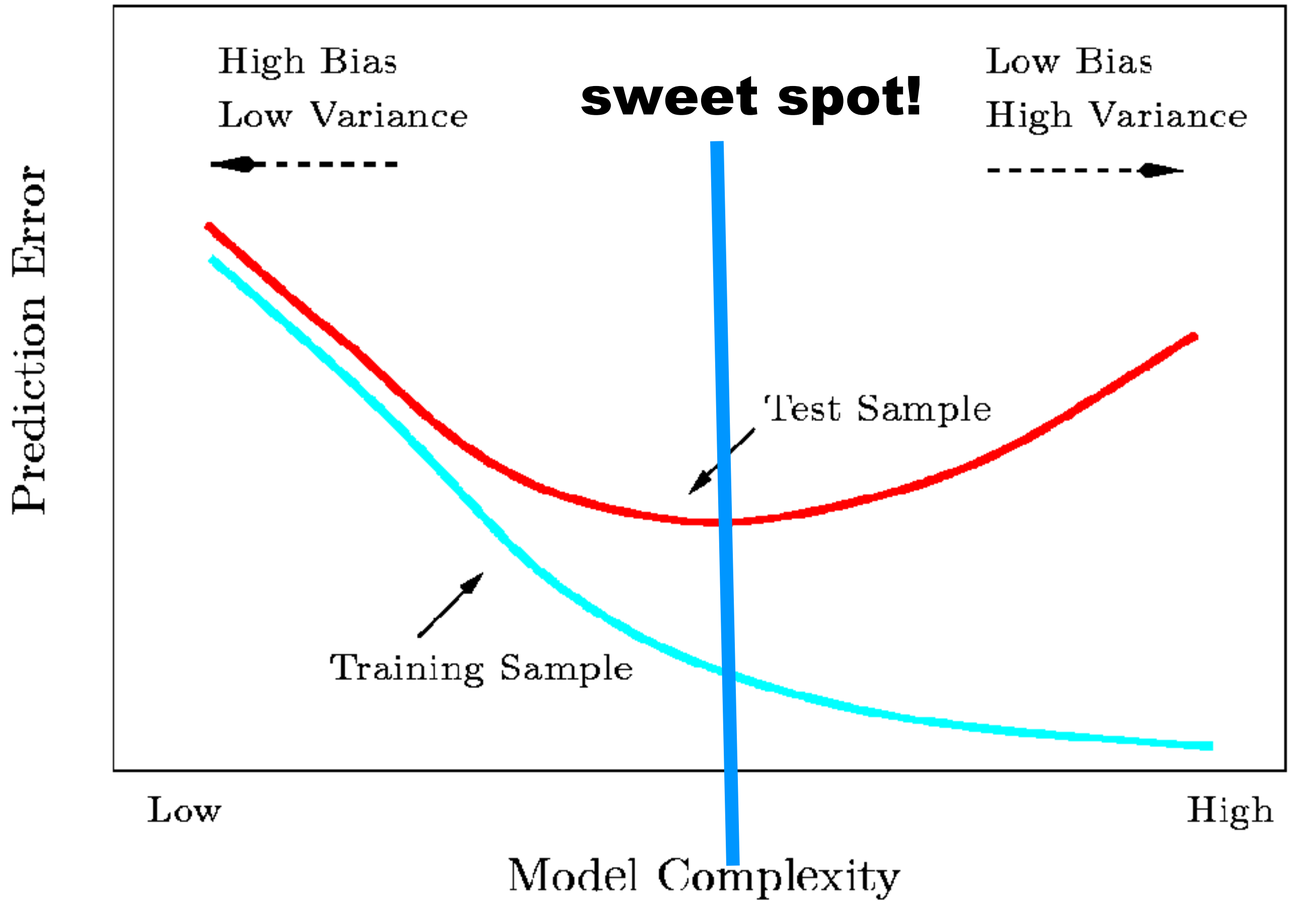
$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$

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

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

λ: "hyperparameter"

Bias-Variance Tradeoff (function of λ)

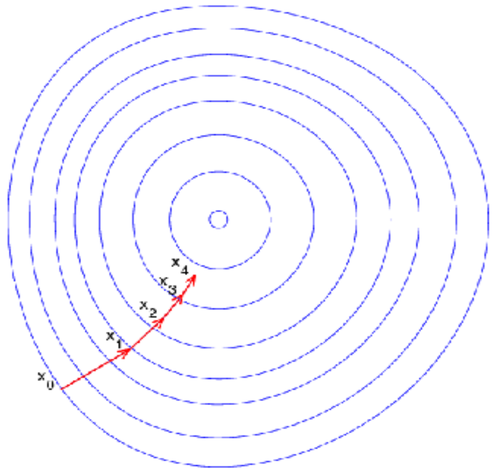


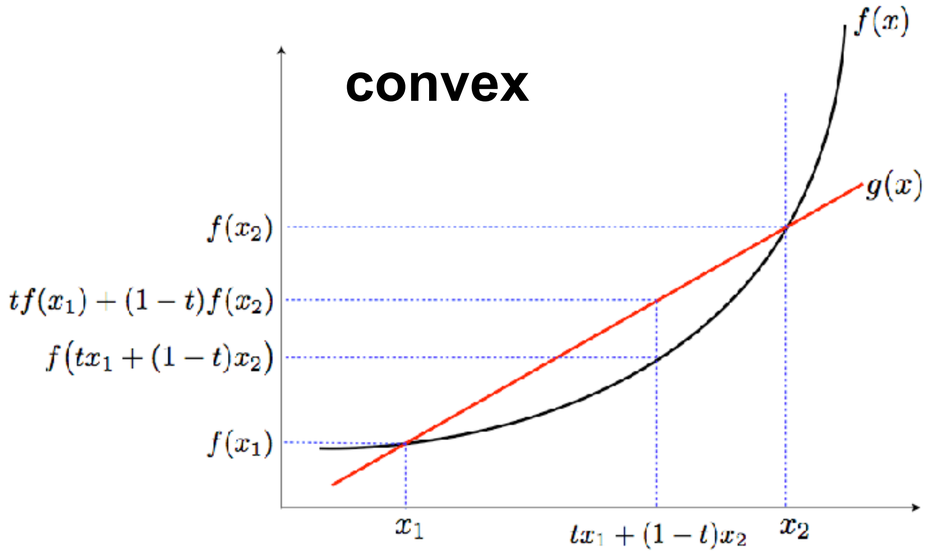
Gradient Descent Minimization

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.





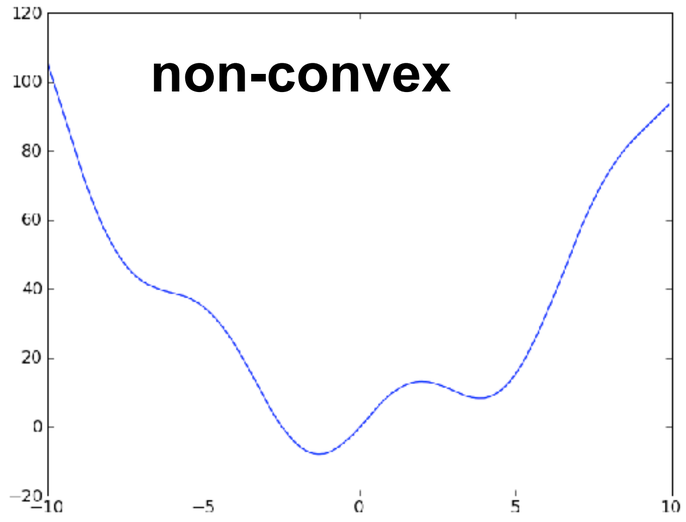


Image Classifier

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

Linear versus Nonlinear Models

- **Objective function #1**

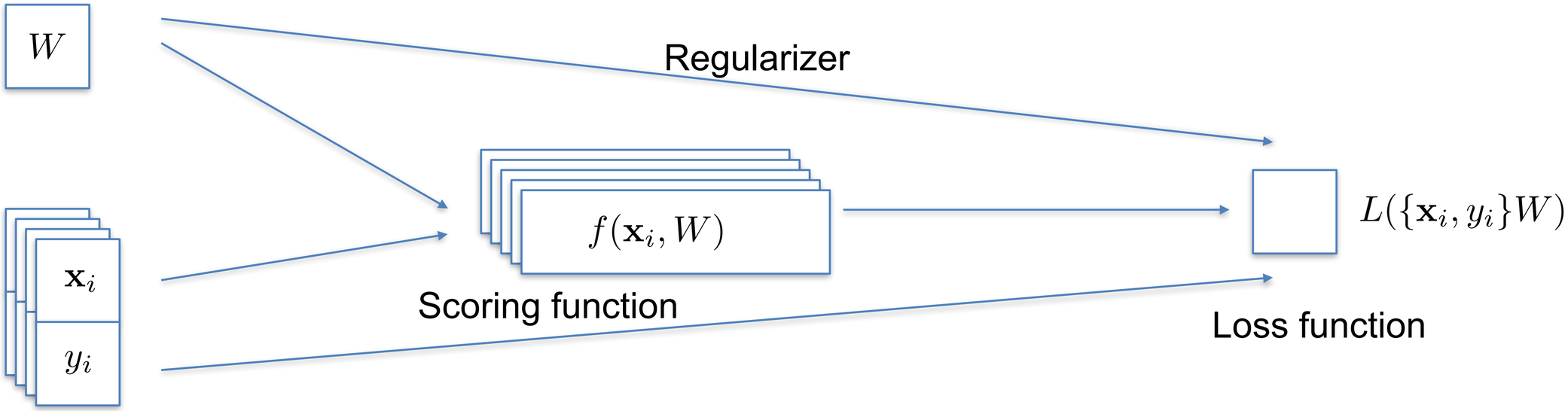
$$s = f(\mathbf{x}_i, W) = W\mathbf{x}_i$$

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

- **Objective function #2**

$$L_i(W) := -\log\left(\frac{e^{Wx_iy_i}}{\sum_j e^{Wx_jy_j}}\right) = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sy_j}}\right)$$

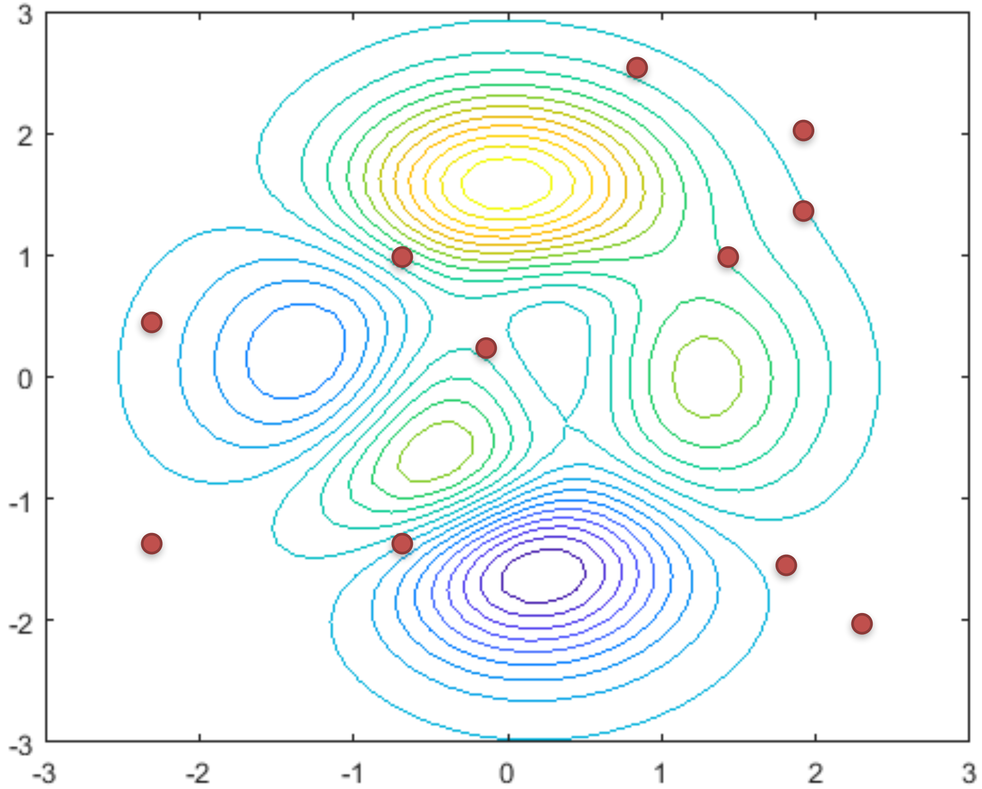
Image Classifier



Method 1a: Random search

- `bestLoss = ∞`
- Loop

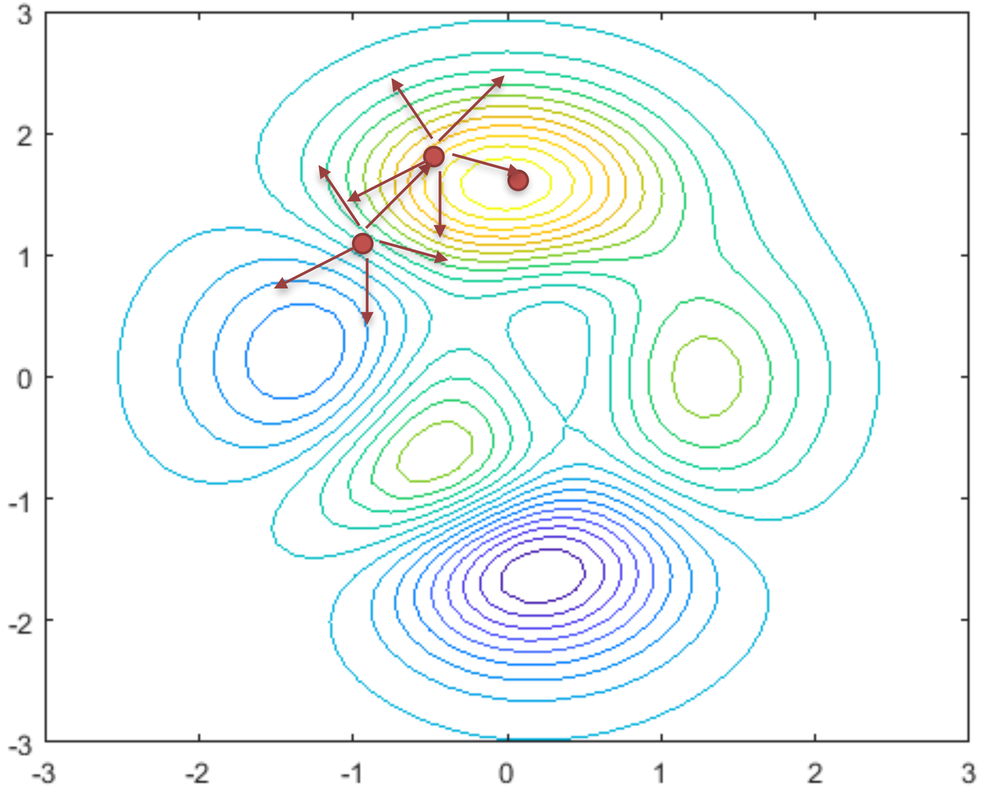
```
        Pick random W
        Compute loss
        if loss<bestLoss
              bestLoss = loss
              currW = W
         endif

  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
```

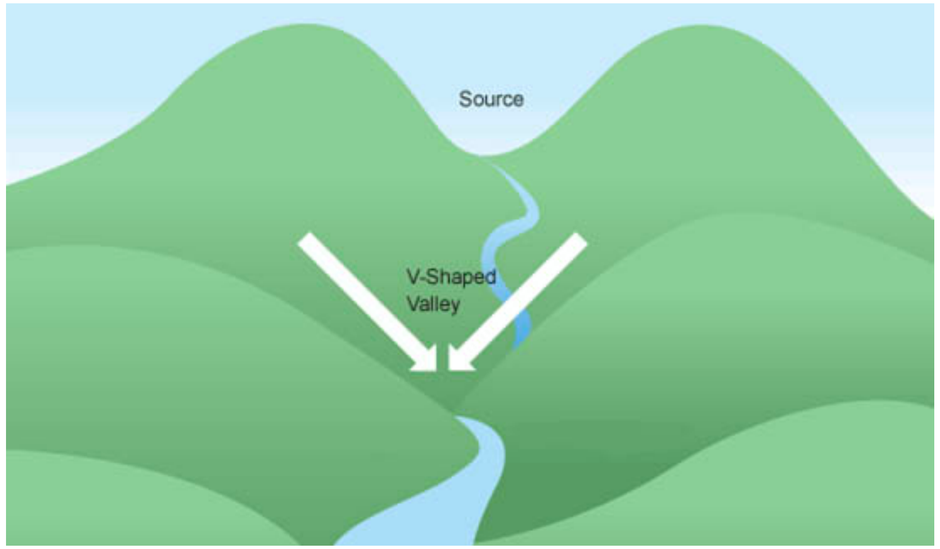


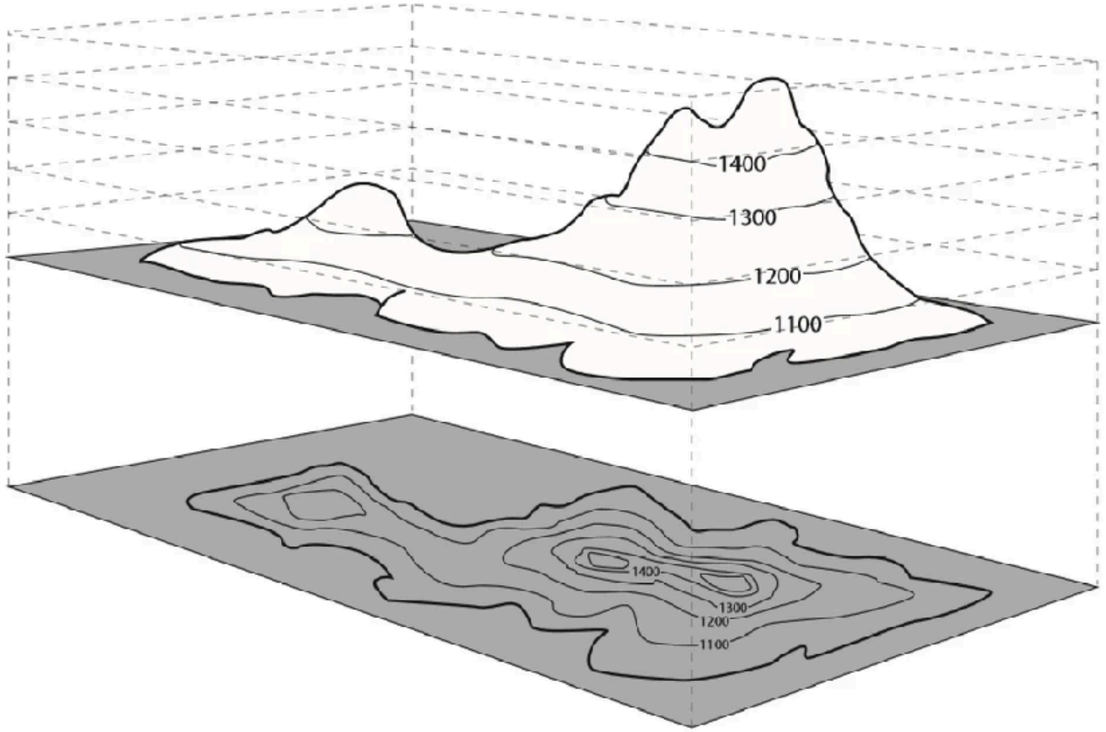
Optimize

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

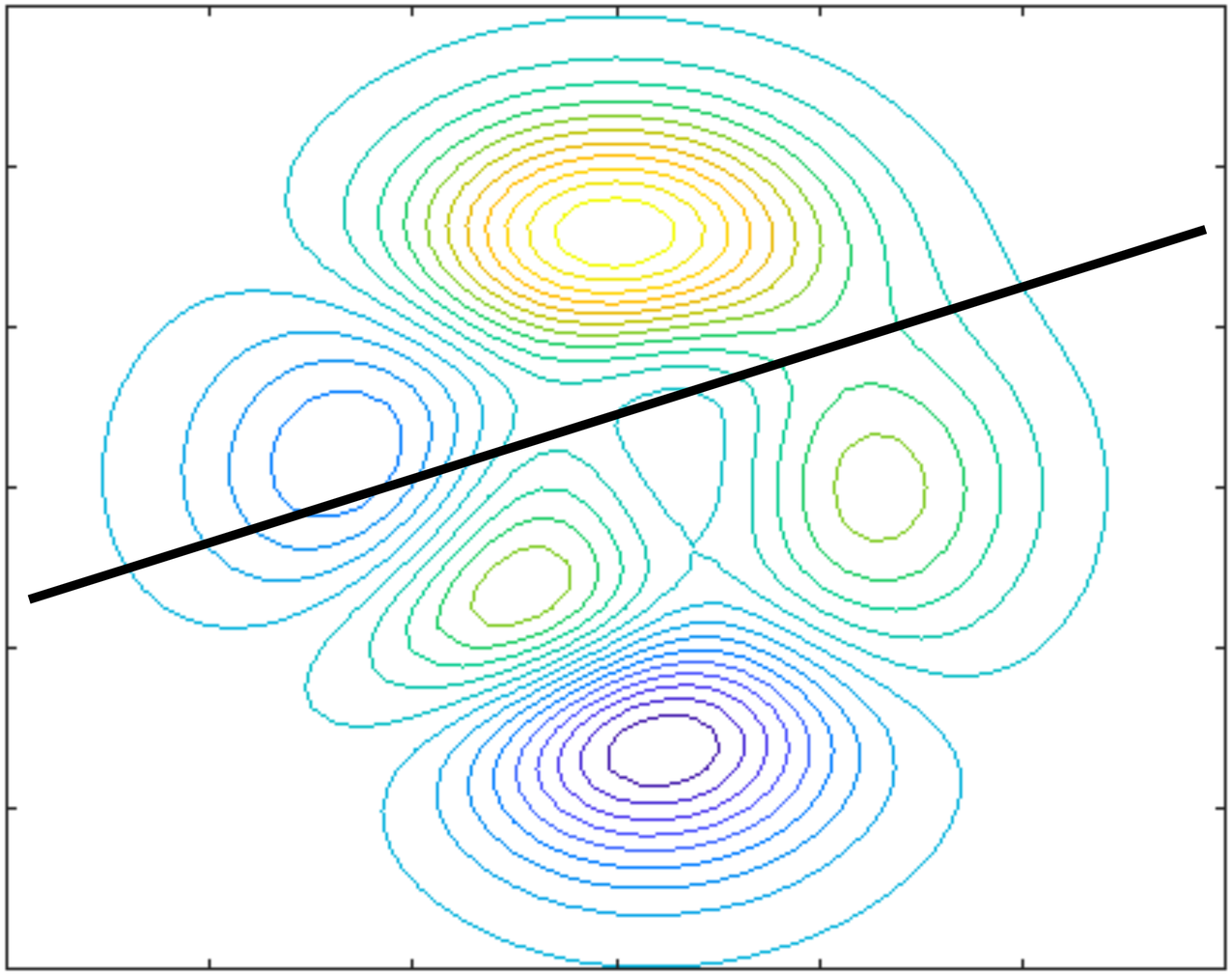
$$\nabla g(\mathbf{x}_i, W) = \left[\frac{\partial g(\mathbf{x}_i, W)}{w_i} = 0 \right]_{D \times 1}$$

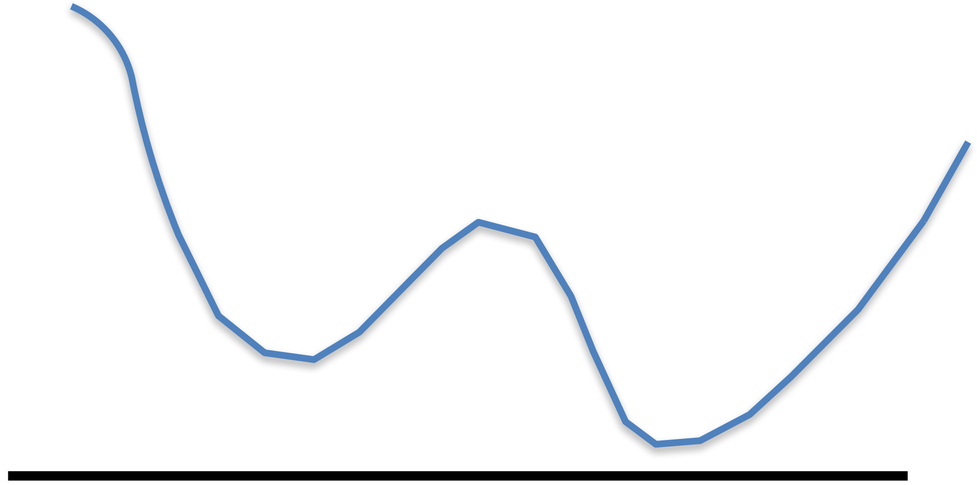
Error Landscape





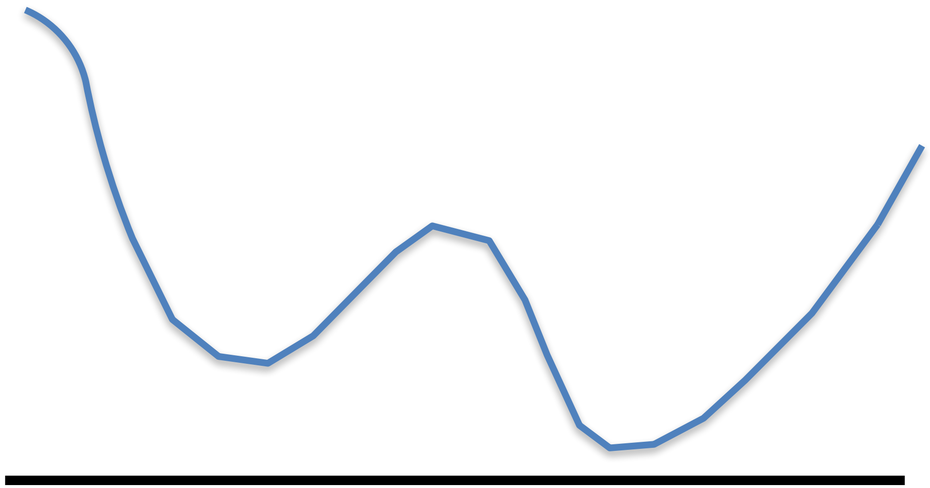
Method 2: Follow Negative of Gradient





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

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

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

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

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

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