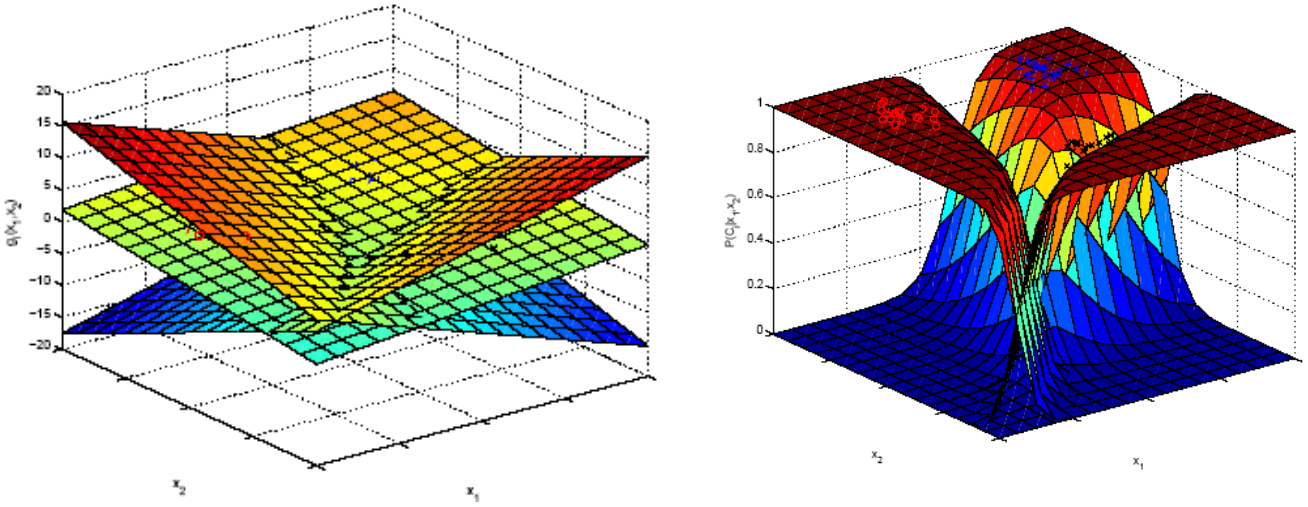
# Introduction to Supervised Learning



Week 3:
Regularization, Logistic Regression

Iasonas Kokkinos

i.kokkinos@cs.ucl.ac.uk

University College London

# Update

References: on moodle

Online resource: David Barber’s book

http://web4.cs.ucl.ac.uk/staff/D.Barber/pmwiki/pmwiki.php?n=Brml.HomePage

# Lecture outline

Recap of week 2

Regularization

Logistic regression

# Questions

Is the loss function appropriate?

- Quadratic loss: convex cost, closed-form solution
- But does the optimized quantity indicate classifier's performance?

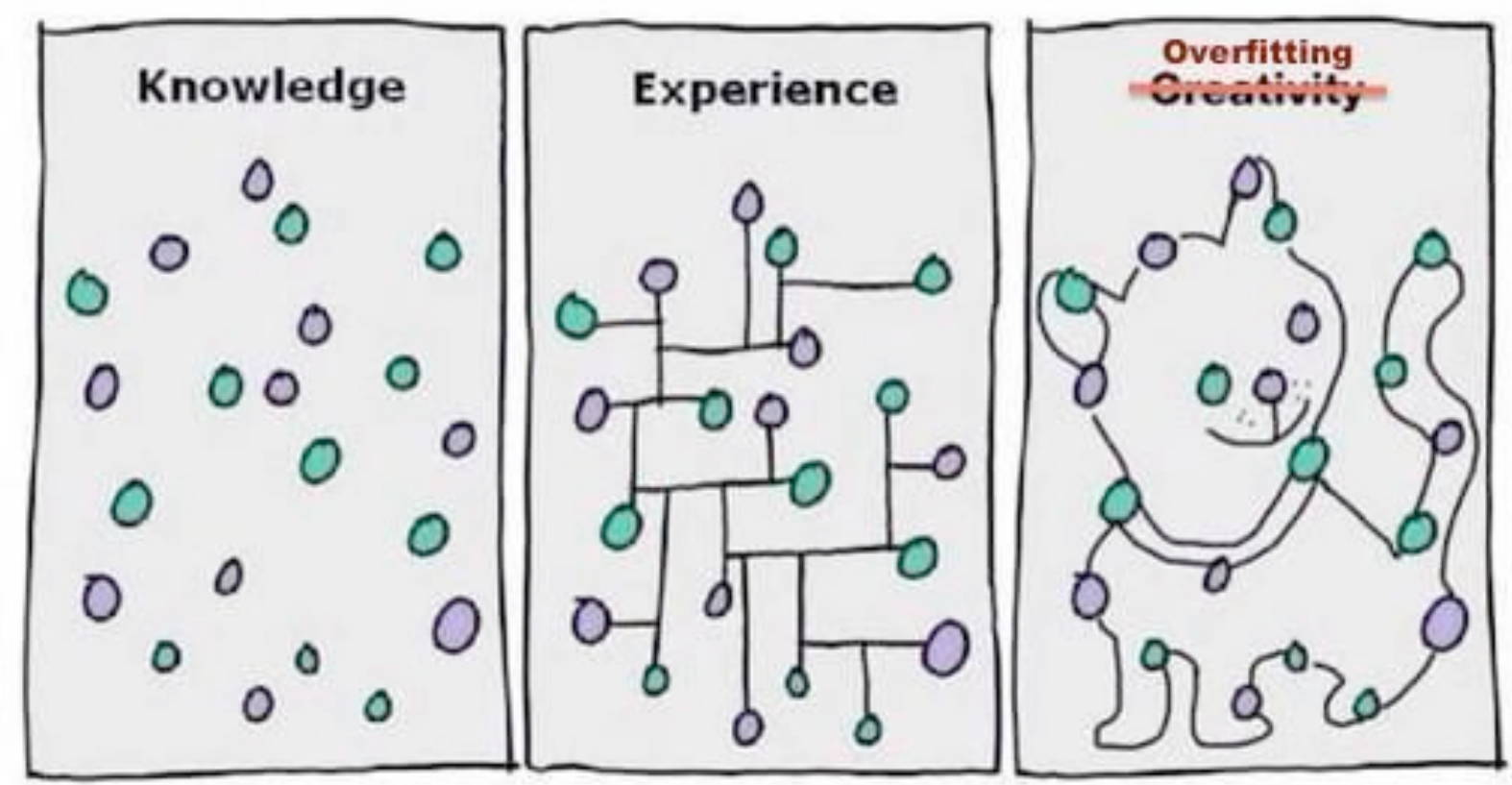
Is the classifier appropriate?

- Linear classifier: fast computation
- But could e.g. a non-linear classifier have better performance?

Are the estimated parameters good?

- Parameters recover input-output mapping on training data
- How can we know they do not simply memorize training data?

# Overfitting problem



# Example: second-order polynomials

$$\mathbf{x} = (x_1, x_2)$$

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ (x_1)^2 \\ (x_2)^2 \\ x_1 x_2 \end{bmatrix}$$

$$\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2$$

# Example: fourth-order polynomials in 5 dimensions

$$\mathbf{x} = (x_1, \ldots, x_5)$$

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_5 \\ \vdots \\ (x_1 x_2 x_3 x_4 x_5)^4 \end{bmatrix}$$

**15625 Dimensions =>15625 parameters**

# What was happening before: approximations

Training: $S = \{(\mathbf{x}^i, y^i)\}, i =,1, \ldots, N$

If **N>D** (e.g. 30 points, 2 dimensions) we have more equations than unknowns: **overdetermined** system!

$$
\begin{aligned}
y^1 &\simeq w_0 x_0^1 + w_1 x_1^1 + \ldots + w_D x_D^1 \\
y^2 &\simeq w_0 x_0^2 + w_1 x_1^2 + \ldots + w_D x_D^2 \\
&\vdots \\
y^N &\simeq w_0 x_0^N + w_1 x_1^N + \ldots + w_D x_D^N
\end{aligned}
$$

Input-output relations can only hold approximately!

# What is happening now: overfitting

Training: $S = \{(\mathbf{x}^i, y^i)\}, i =,1, \ldots, N$

If **N<D** (e.g. 30 points, 15265 dimensions) we have more unknowns than equations: **underdetermined** system!

$$y^1 = w_0 x_0^1 + w_1 x_1^1 + \ldots + w_D x_D^1$$
$$y^2 = w_0 x_0^2 + w_1 x_1^2 + \ldots + w_D x_D^2$$
$$\vdots$$
$$y^N = w_0 x_0^N + w_1 x_1^N + \ldots + w_D x_D^N$$

Input-output equations hold exactly, but we are simply memorizing data

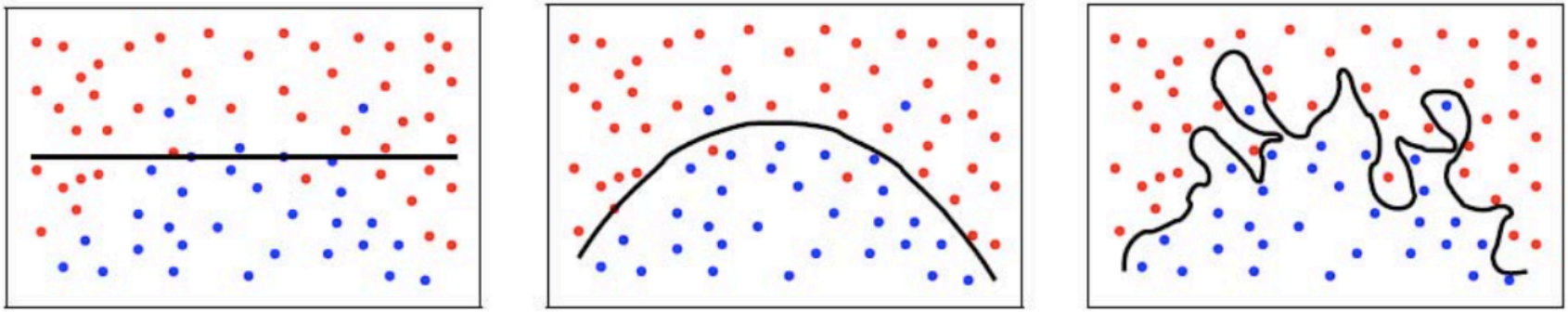
# Overfitting, in images

## Classification

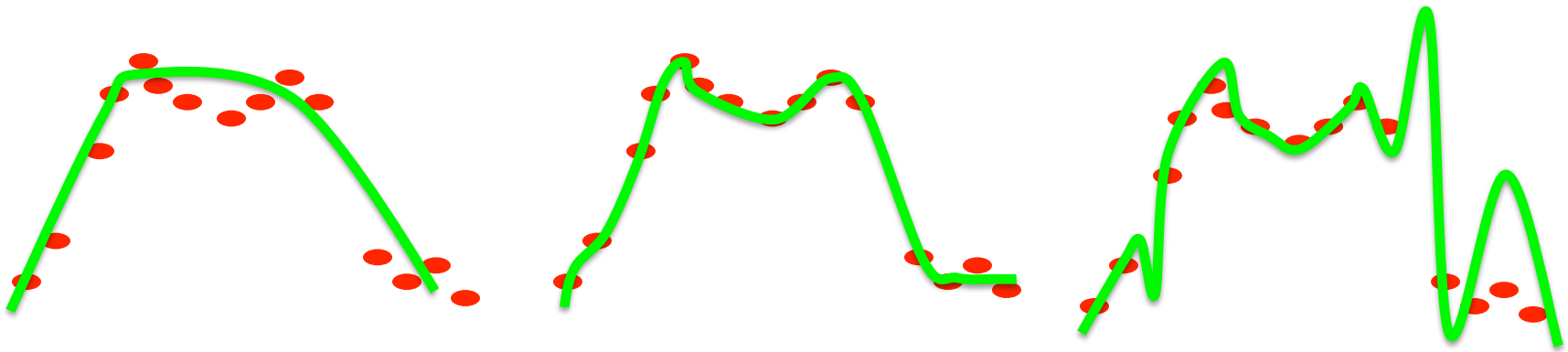
Underfitting

just right ⟷

Overfitting



## Regression



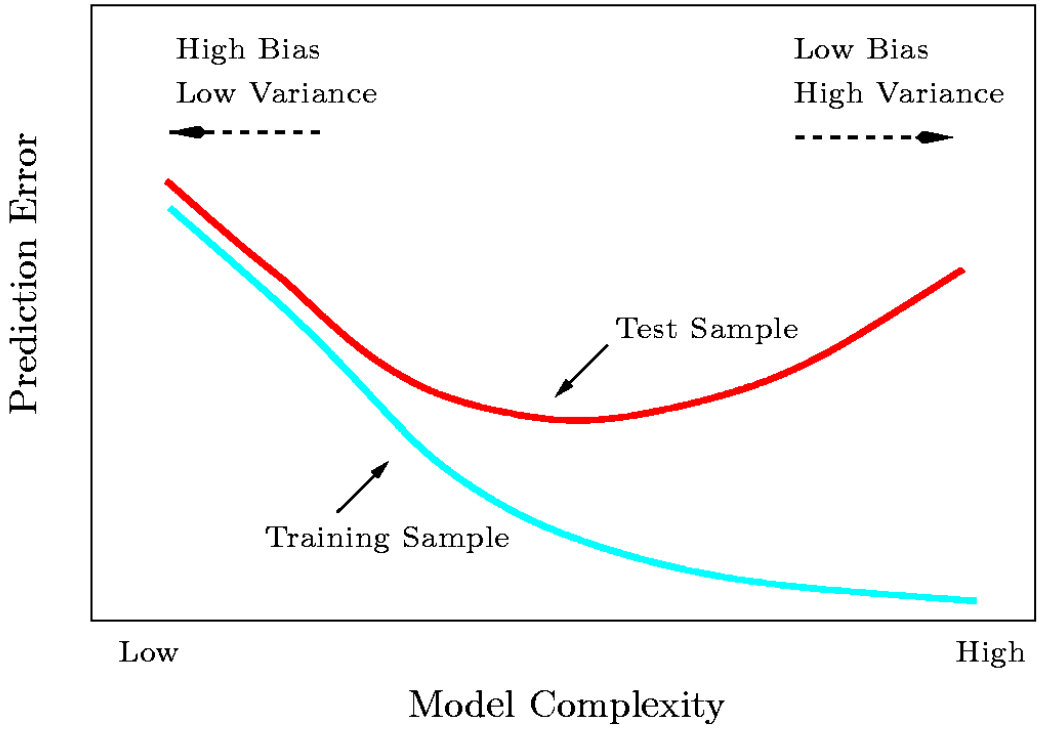
# Tuning the model's complexity

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

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

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

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



# Lecture outline

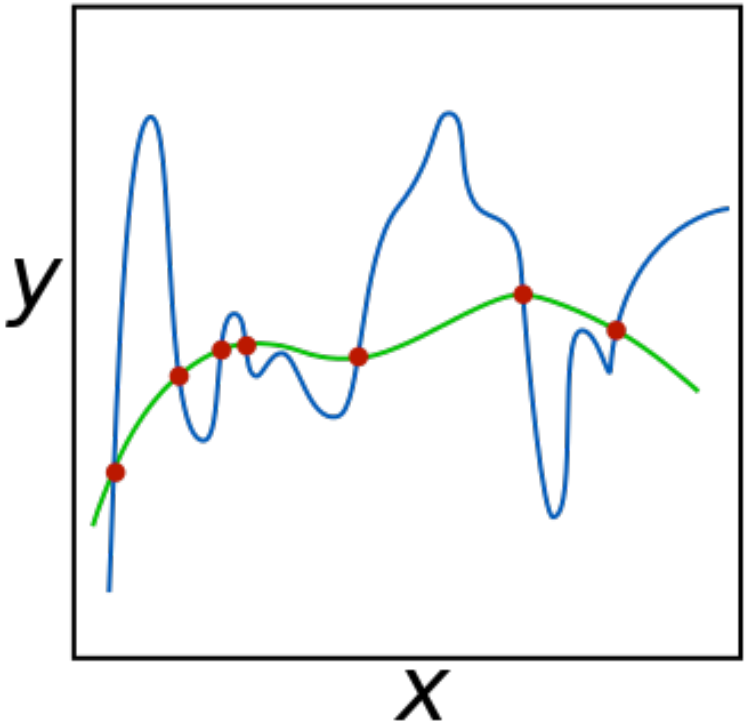
Recap of week 2

Regularization

Logistic regression

# Regularization: keeping it simple

In high dimensions: too many solutions for the same problem




Regularization: prefer the least complex among them

How? Penalize complexity

# How to control complexity?

Observation: problem started with high-dimensional embeddings

Guess: Number of dimensions relates to “complexity”

(Week 4: we will guess again!)

Intuition: with many parameters, we can fit anything

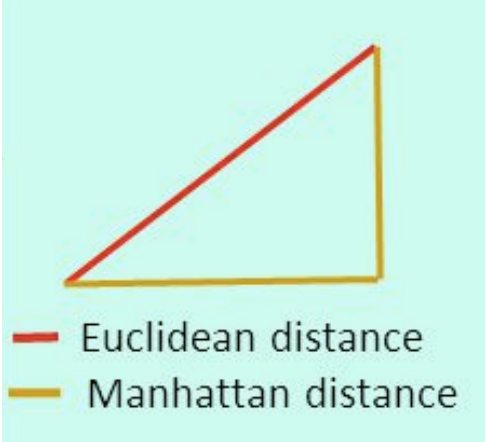
But what if we force the classifier not to use all of the parameters?

Idea: penalize the use of large parameter values

**How do we measure “large”?**

**How do we enforce small values?**

# How do we measure “large”?




Method parameters: D-dimensional vector

$$\mathbf{w} = [w_1, w_2, \ldots, w_D]$$

“Large” vector: vector **norm**

L2, (“euclidean”) norm:

$$\|\mathbf{w}\|_2 \doteq \sqrt{\sum_{d=1}^{D} w_d^2} = \sqrt{\langle \mathbf{w}, \mathbf{w} \rangle}$$

L1, (“manhattan”) norm:

$$\|\mathbf{w}\|_1 \doteq \sum_{d=1}^{D} |w_d|$$

Lp norm, p>1:

$$\|\mathbf{w}\|_p \doteq \left( \sum_{d=1}^{D} w_d^p \right)^{1/p}$$

# Linear vs. Ridge 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) $R(\mathbf{w}) \doteq \|\mathbf{w}\|_2^2 = \mathbf{w}^T\mathbf{w}$

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

minimum remains to be determined

"data fidelity"

complexity

scalar, remains to be determined

# Least squares solution (week 2)

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

**Condition for minimum:**

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

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

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

# Ridge regression: L2-regularized linear regression

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

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

as before, for linear regression

identity matrix

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

**Condition for minimum:**

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

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

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

# Ridge regression, continued

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

New objective:

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

"data fidelity"

complexity

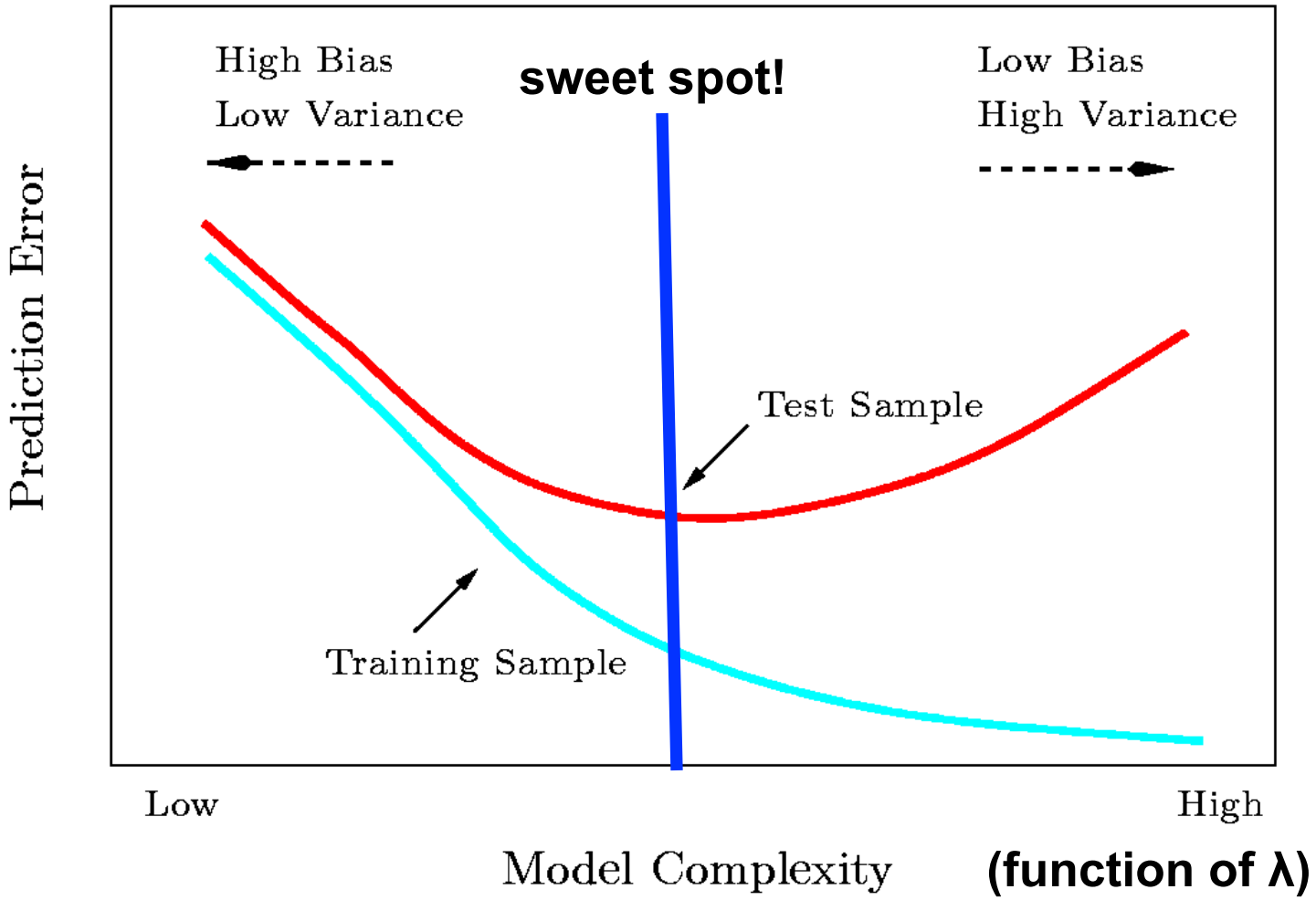
We just determined minimum

scalar, remains to be determined

λ: "hyperparameter"

**Νοτε: δirect minimization w.r.t. it would lead to λ=0**

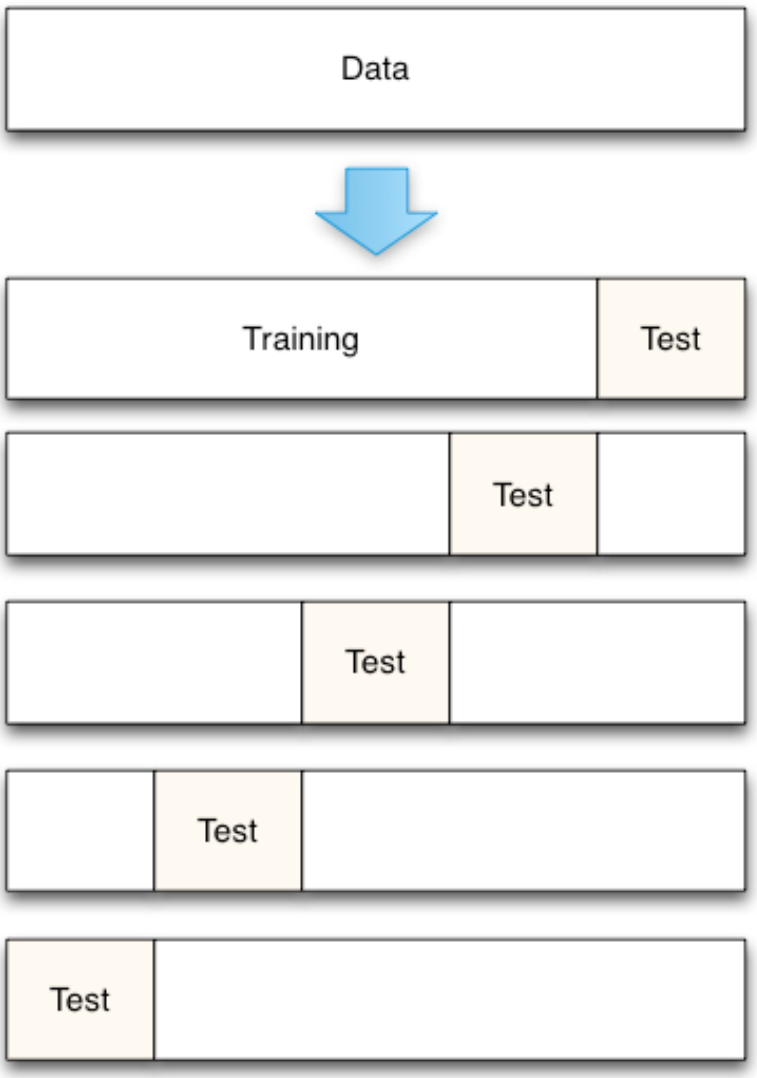
# Bias-Variance tradeoff as a function of λ



# Selecting λ with cross-validation

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

**Least glorious, most effective of all methods**



# Next week's assignment

**K-fold cross-validation procedure (1.8/2.3)**

For every value of $\lambda$:

- For every round $k \in \{1, \ldots, K\}$:
  - a Split the training data into a parameter estimation set, $E_k$, and a validation set, $V_k$.
  - b Find $w_{o,k}$ using by minimizing $E(f, E_k) + \lambda \|w\|^2$.
  - c Estimate the associated loss, $E_{k,\lambda,o} = \sum_{i \in V_k} [y^i \neq \text{sign}(\langle w, x^i \rangle)]$ on the 'validation'
- Average the loss over the $K$ validation rounds to find the cross-validation loss associated with a particular $\lambda$:

$$L_{o,\lambda} = \frac{1}{K} \sum_{k=1}^{K} E_{k,\lambda,o} \tag{2}$$

Identify the value of $\lambda_o^* = \arg\min_\lambda L_{o,\lambda}$ that gives the lowest cross-validation loss.
Obtain $w_o$ as the solution of $E(f, \mathcal{S}) + \lambda_o^* \|w\|^2$, namely use the optimal $\lambda$ and the whole training set.

Include as part of your report

# Lecture outline

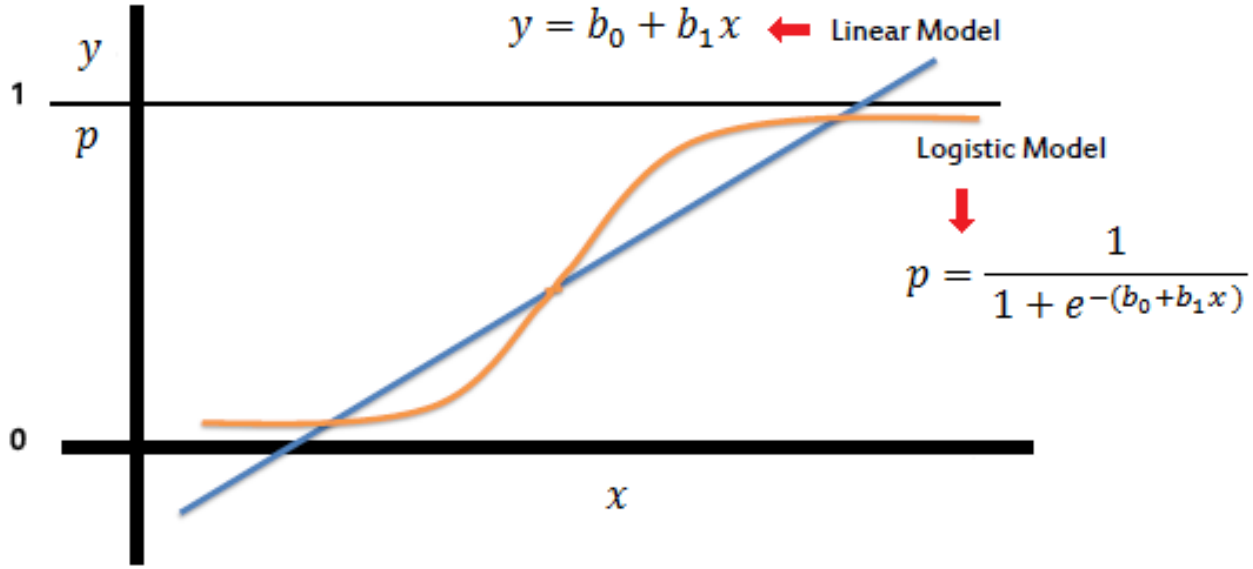
Regularization

Logistic Regression

Training criterion formulation

Interpretation

Optimization



# Questions

Is the loss function appropriate?

- Quadratic loss: convex cost, closed-form solution
- But does the optimized quantity indicate classifier's performance?

Is the classifier appropriate?

- Linear classifier: fast computation
- But could e.g. a non-linear classifier have better performance?

Are the estimated parameters good?

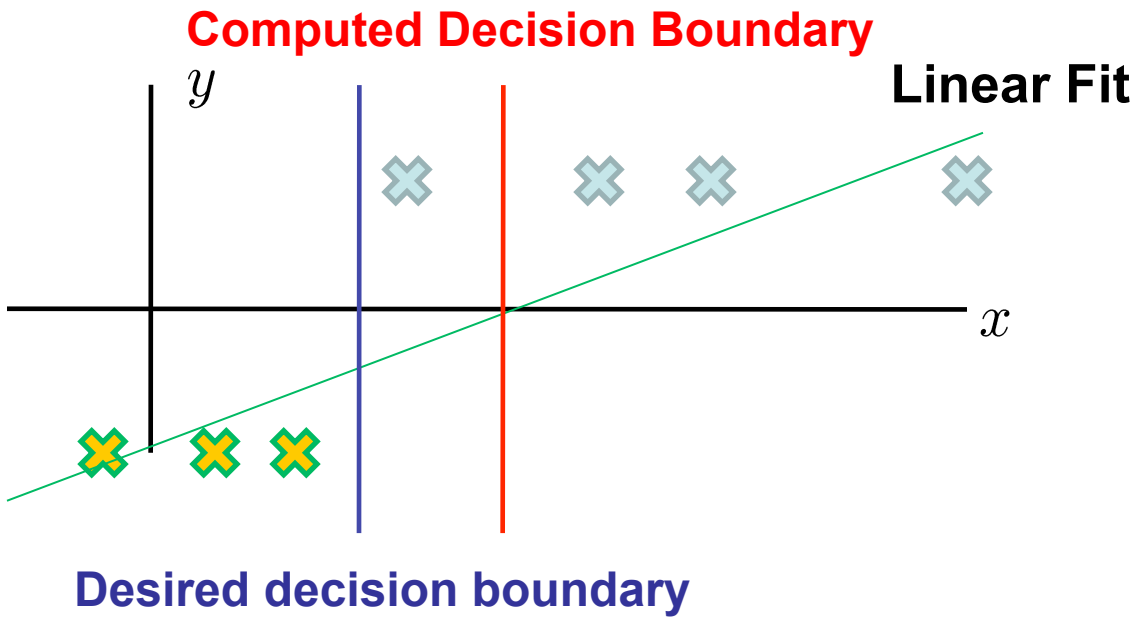
- Parameters recover input-output mapping on training data
- How can we know they do not simply memorize training data?

# Inappropriateness of quadratic loss (last week)

We chose the quadratic cost function for convenience

Single, global minimum & closed form expression

But does it indicate classification performance?

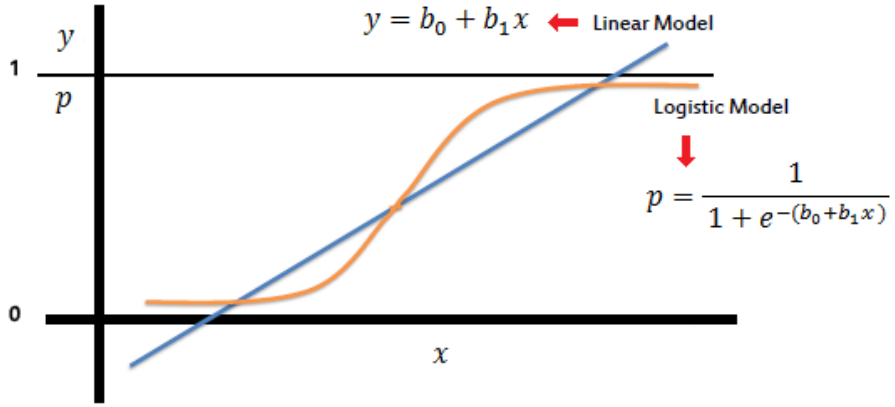


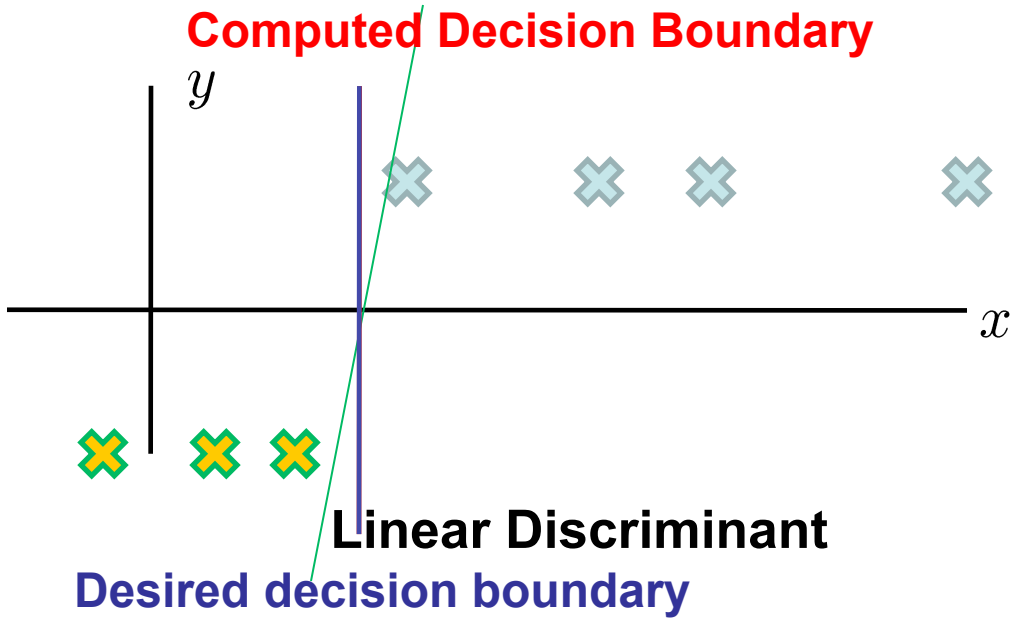
# Basic idea

Use squashing function

The higher, the better

We will make this (very) precise

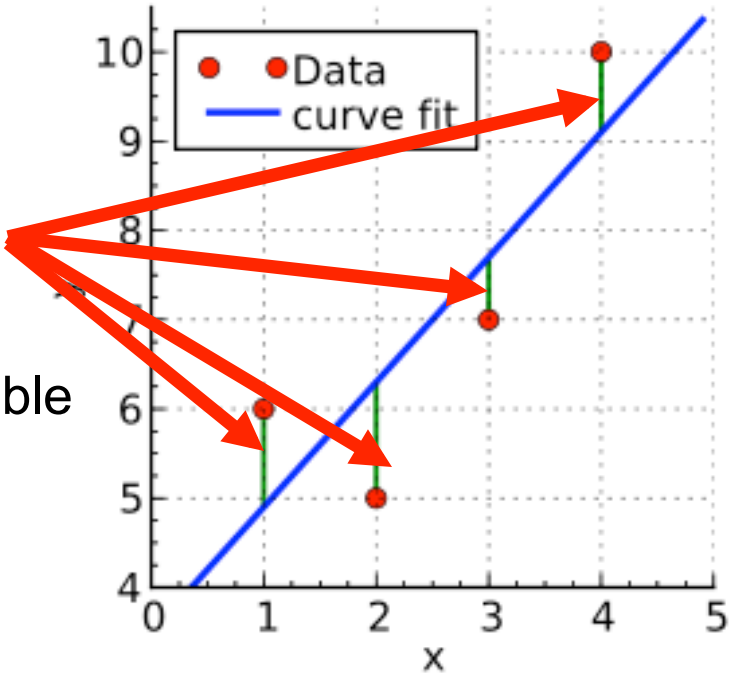


# Probabilistic formulation of linear regression-I

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

**residuals**




Residual: zero-mean gaussian random variable

$$E^i \sim N(0, \sigma^2)$$

$$p(\epsilon^i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(e^i)^2}{2\sigma^2}\right)$$

Conditional model on observations (using $\epsilon^i = y^i - \mathbf{w}^T \mathbf{x}^i$)

$$p(y^i|\mathbf{x}^i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^i - \mathbf{w}^T \mathbf{x}^i)^2}{2\sigma^2}\right)$$

# Probabilistic interpretation of linear regression

**Training set:** $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}, \mathbf{x} \in \mathbb{R}^D, y \in \mathbb{R}$

$$p(y^1, \ldots, y^n | \mathbf{x}^1, \ldots, \mathbf{x}^N) =$$

**Independence assumption**

$$= \prod_{i=1}^{N} p(y^i | \mathbf{x}^i)$$

$$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^i - \mathbf{w}^T \mathbf{x}^i)^2}{2\sigma^2}\right)$$

$$= \frac{1}{(\sqrt{2\pi}\sigma)^N} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{N} (y^i - \mathbf{w}^T \mathbf{x}^i)^2\right)$$

**=> Least squares: Maximum Likelihood estimation of w**

# Great, but only for real-valued outputs!

**Training set:** $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}, \mathbf{x} \in \mathbb{R}^D, y \in \mathbb{R}$

$$p(y^1, \ldots, y^n | \mathbf{x}^1, \ldots, \mathbf{x}^N) =$$

**Independence assumption**

$$= \prod_{i=1}^{N} p(y^i | \mathbf{x}^i)$$

$$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^i - \mathbf{w}^T \mathbf{x}^i)^2}{2\sigma^2}\right)$$

$$= \frac{1}{(\sqrt{2\pi}\sigma)^N} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{N} (y^i - \mathbf{w}^T \mathbf{x}^i)^2\right)$$

# From regression to classification

**Training set:** $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}, \mathbf{x} \in \mathbb{R}^D, y \in \{0, 1\}$

$$p(y^1, \ldots, y^n | \mathbf{x}^1, \ldots, \mathbf{x}^N) =$$

**Independence assumption**

$$= \prod_{i=1}^{N} p(y^i | \mathbf{x}^i)$$

?

# Bernoulli distribution

Discrete random variable $Y \in \{0, 1\}$

1x2 table, 1 parameter:

$$P(Y = 1) = p$$
$$P(Y = 0) = 1 - P(Y = 1) = 1 - p$$

Compact form:

$$P(Y = c) = \begin{cases} p & c = 1 \\ 1 - p & c = 0 \end{cases}$$
$$= p^c (1 - p)^{1-c}$$

# Parametric model for posterior

$$P(Y = 1|X = \mathbf{x}; \mathbf{w}) = f(\mathbf{x}, \mathbf{w})$$
$$P(Y = 0|X = \mathbf{x}; \mathbf{w}) = 1 - f(\mathbf{x}, \mathbf{w})$$

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

# Form of posterior distribution

Week2: Bayes' rule, 2 Gaussians: $P(Y=1|X=\mathbf{x}) = \dfrac{1}{1+\exp(\mathbf{b}^T\mathbf{x}+c)}$

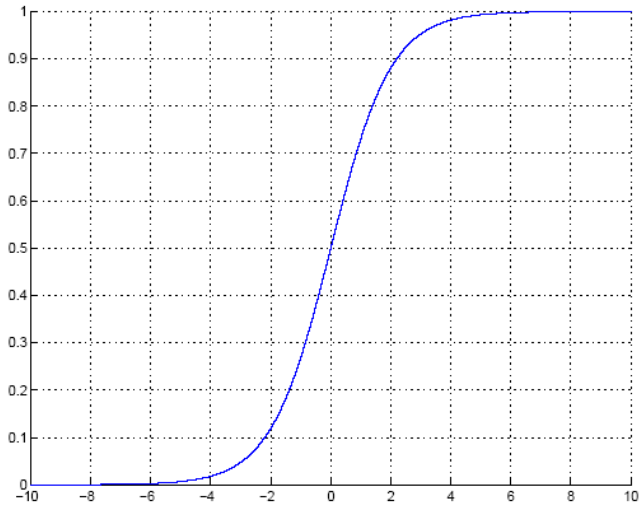
$$\mathbf{b} = \Sigma^{-1}(\mu_0 - \mu_1) \quad c = \frac{1}{2}\left[\mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0\right] + \log\left(\frac{1-\pi}{\pi}\right)$$

Now: keep the form, drop the Bayes-based derivation

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

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

"squashing function": $-\infty \to 0$, $+\infty \to 1$



# From regression to classification

**Training set:** $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}, \mathbf{x} \in \mathbb{R}^D, y \in \{0, 1\}$

$$p(y^1, \ldots, y^n | \mathbf{x}^1, \ldots, \mathbf{x}^N) =$$

**Independence assumption**

$$= \prod_{i=1}^{N} P(y^i | \mathbf{x}^i)$$

?

# From regression to classification, continued

**Training set:** $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}, \mathbf{x} \in \mathbb{R}^D, y \in \{0, 1\}$

$$p(y^1, \ldots, y^n | \mathbf{x}^1, \ldots, \mathbf{x}^N) =$$

**Independence assumption**

$$= \prod_{i=1}^{N} p(y^i | \mathbf{x}^i)$$

$$= \prod_{i=1}^{N} g(\mathbf{w}^T \mathbf{x}^i)^{y^i} (1 - g(\mathbf{w}^T \mathbf{x}^i))^{1-y^i}$$

$$\log P(\mathbf{y}|\mathbf{X}; \mathbf{w}) = \sum_{i=1}^{N} \log \left( g(\mathbf{w}^T \mathbf{x}^i)^{y^i} \right) + \log \left( \left(1 - g(\mathbf{w}^T \mathbf{x}^i)\right)^{(1-y^i)} \right)$$

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

**Q1: How does this behave?**

Q2: How to optimize it with respect to w?

# Loss function for linear regression

Training: given $S = \{(\mathbf{x}^i, y^i)\}, i = 1, \ldots, N$, estimate optimal $\mathbf{w}$

Loss function: quantify appropriateness of $\mathbf{w}$

$$L(S, \mathbf{w}) = \sum_{i=1}^{N} l(y^i, f_{\mathbf{w}}(\mathbf{x}^i))$$

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

# Loss function for classification

Training: given $S = \{(\mathbf{x}^i, y^i)\}, i = 1, \ldots, N$, estimate optimal $\mathbf{w}$

Loss function: quantify appropriateness of $\mathbf{w}$

$$
\begin{aligned}
L(S, \mathbf{w}) &= -\log P(\mathbf{y}|\mathbf{X}; \mathbf{w}) \\
&= -\sum_{i=1}^{N} \log P(y^i|\mathbf{x}^I; \mathbf{w}) \\
&= -\sum_{i=1}^{N} y^i \log g(\mathbf{w}^T \mathbf{x}^i) + (1 - y^i) \log(1 - g(\mathbf{w}^T \mathbf{x}^i)) \\
&= \sum_{i=1}^{N} l(y^i, f_{\mathbf{w}}(\mathbf{x}^i))
\end{aligned}
$$

**Linear discriminant:** $f_{\mathbf{w}}(\mathbf{x}^i) = \mathbf{w}^T \mathbf{x}^i$

# Rewriting the quadratic loss

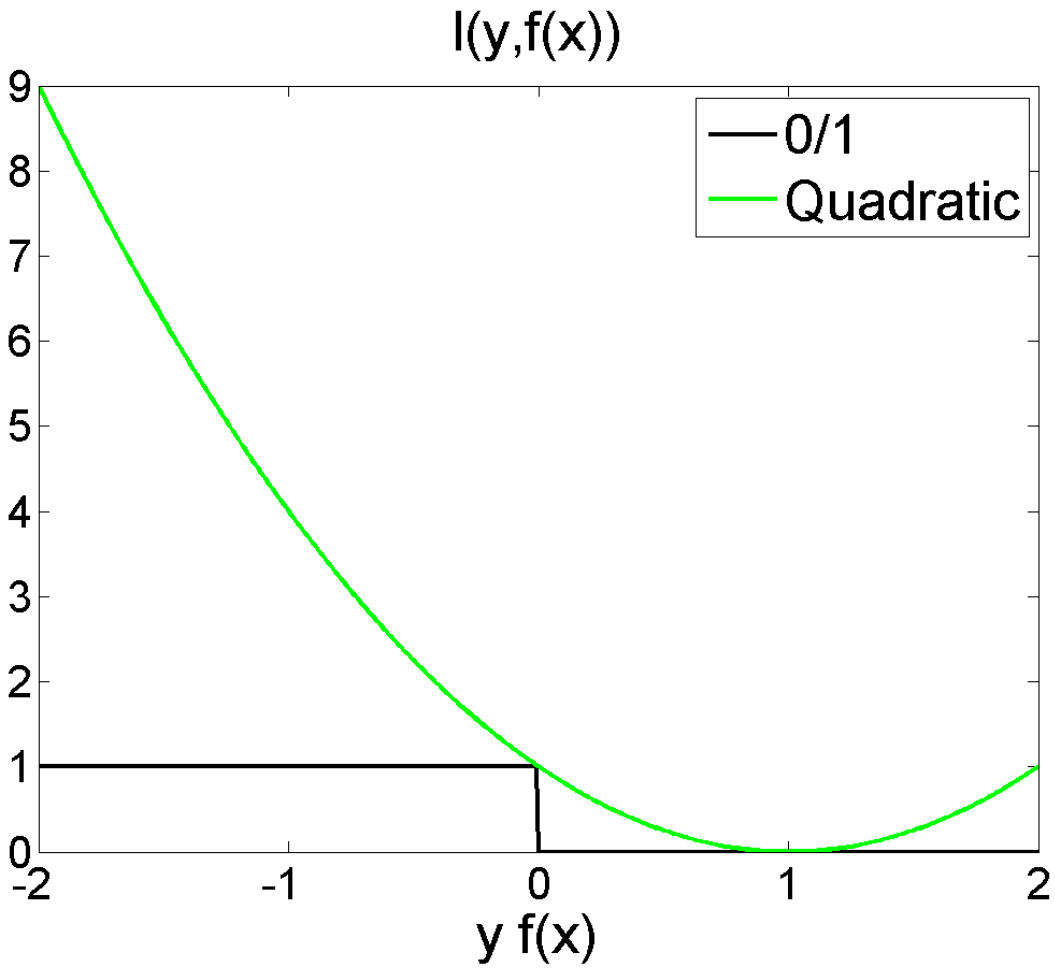
Consider transformation: $y_{\pm} = 2y_b - 1$

$y_b \in \{0, 1\} \qquad y_{\pm} \in \{-1, 1\}$

$$
\begin{aligned}
l(y, f_{\mathbf{w}}(\mathbf{x})) &= (y - f_{\mathbf{w}}(\mathbf{x}))^2 \\
&\overset{y^2=1}{=} y^2(y - f_{\mathbf{w}}(\mathbf{x}))^2 \\
&= (y^2 - y f_{\mathbf{w}}(\mathbf{x}))^2 \\
&\overset{y^2=1}{=} (1 - y f_{\mathbf{w}}(\mathbf{x}))^2
\end{aligned}
$$

# Inappropriateness of quadratic loss for classification

Last slide: $l(y, f(x)) = (1 - yf(x))^2$




Quadratic loss is not robust to outliers and penalizes outputs that are `too good'

# Rewriting the cross-entropy loss

As before: $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

$y_{\pm} = 2y_b - 1 \quad y_b \in \{0, 1\}$

$y_{\pm} \in \{-1, 1\}$

$$P(Y = 1|X = \mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-h_{\mathbf{w}}(\mathbf{x}))}$$

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

$$= 1 - \frac{1}{1 + \exp(-h_{\mathbf{w}}(\mathbf{x}))}$$

$$= \frac{\exp(-h_{\mathbf{w}}(\mathbf{x}))}{1 + \exp(-h_{\mathbf{w}}(\mathbf{x}))}$$

$$= \frac{1}{1 + \exp(h_{\mathbf{w}}(\mathbf{x}))}$$

# Rewriting the cross-entropy loss

As before: $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ $\quad y_{\pm} = 2y_b - 1$ $\quad y_b \in \{0, 1\}$

$y_{\pm} \in \{-1, 1\}$

**Last slide:**

$$P(Y = 1|X = \mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-h_{\mathbf{w}}(\mathbf{x}))}$$

$$P(Y = -1|X = \mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(h_{\mathbf{w}}(\mathbf{x}))}$$

**Compact form:**

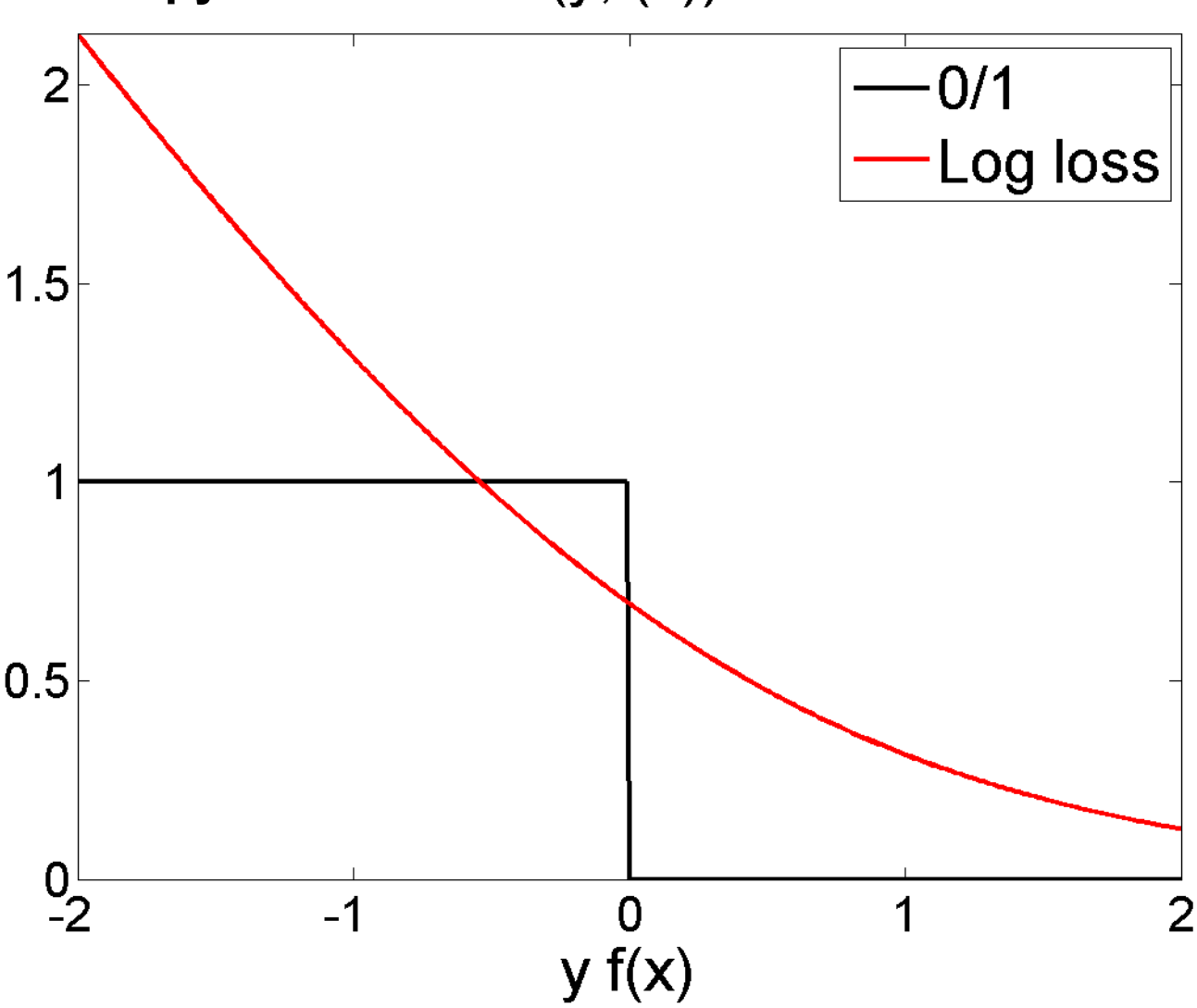
$$P(Y = y|X = \mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-yh_{\mathbf{w}}(\mathbf{x}))}$$

$$L(S, \mathbf{w}) = \sum_{i=1}^{N} -\log P(Y = y^i|X = \mathbf{x}^i; \mathbf{w})$$

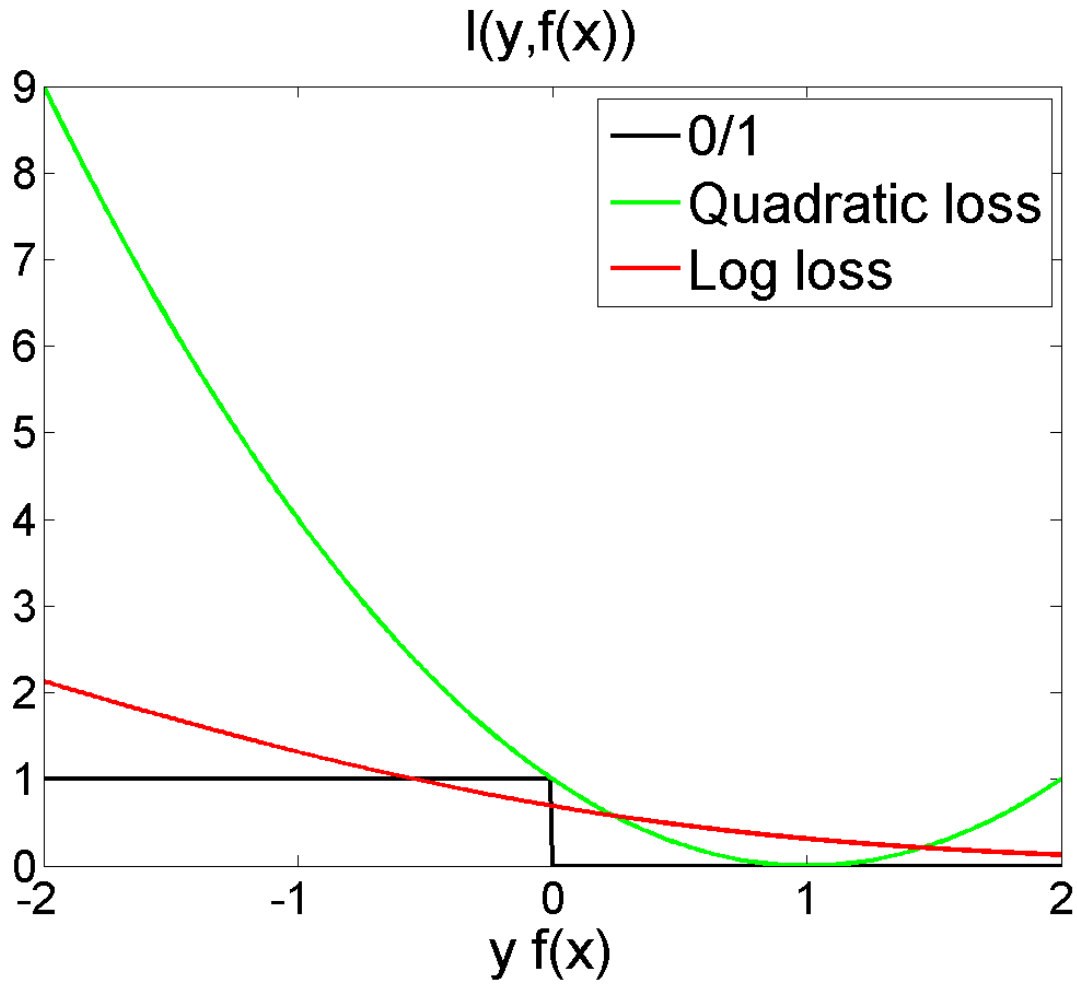
$$= \sum_{i=1}^{N} \log(1 + \exp(-y^i h_{\mathbf{w}}(\mathbf{x}^i)))$$

**Log loss:** $l(y, f(x)) = \log(1 + \exp(-yf(x)))$

**a.k.a. "cross entropy" loss**



# Log loss vs. quadratic loss




Quadratic loss

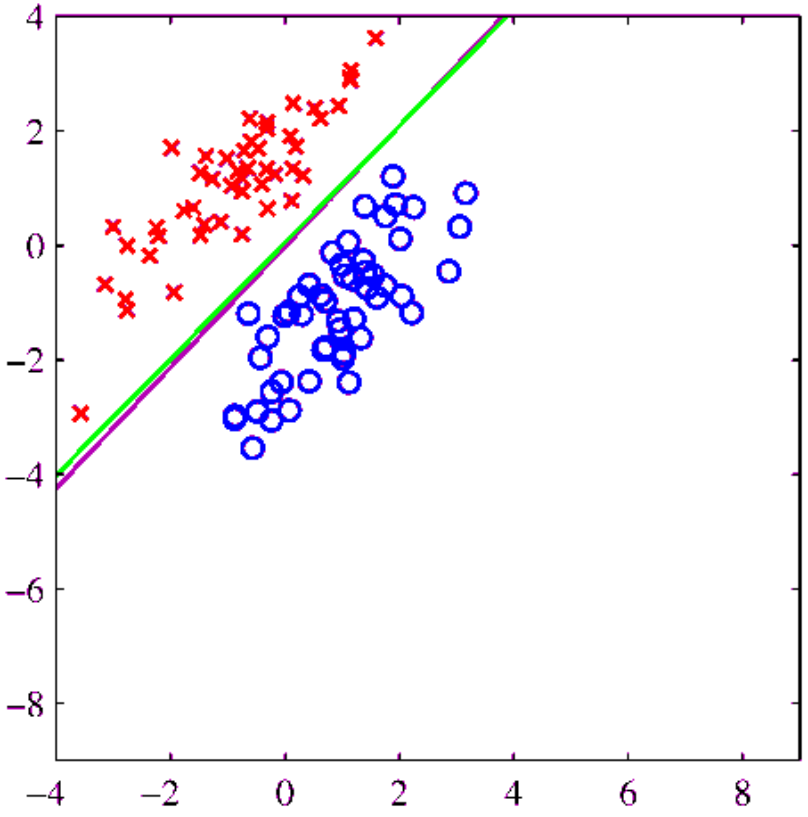
$$l(y, f(x)) = (1 - yf(x))^2$$

Log loss

$$l(y, f(x)) = \log(1 + \exp(-yf(x)))$$

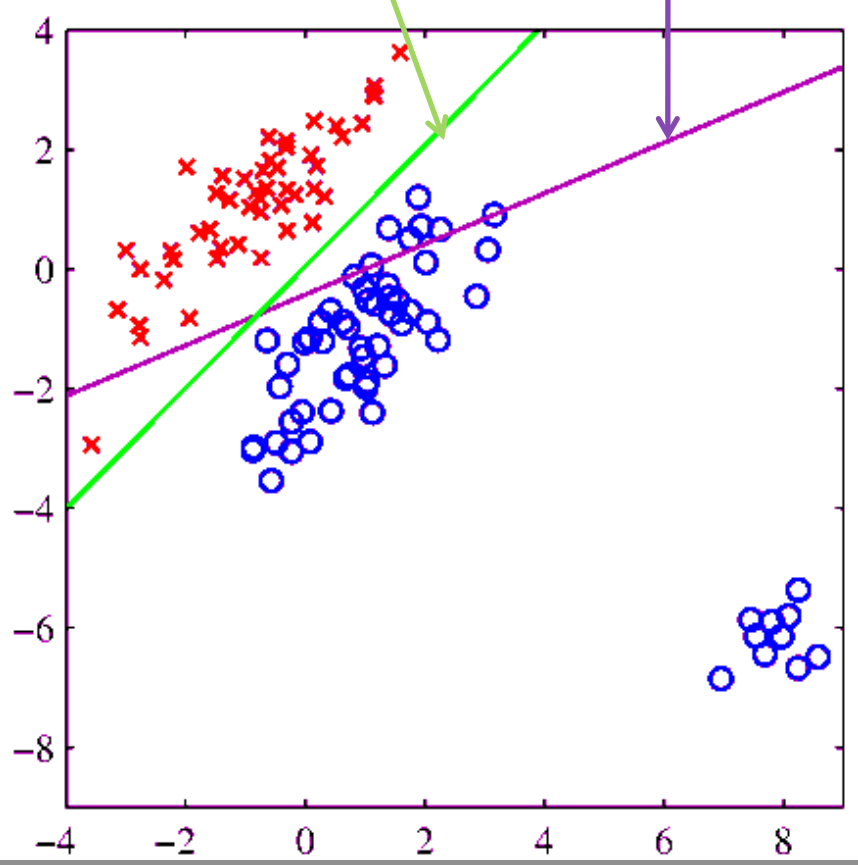
# Logistic vs Linear Regression

Logistic regression is more robust




Linear Regression

Logistic Regression



# Questions

Is the loss function appropriate?

- Quadratic loss: convex cost, closed-form solution
- But does the optimized quantity indicate classifier's performance?

Is the classifier appropriate?

- Linear classifier: fast computation
- But could e.g. a non-linear classifier have better performance?

Are the estimated parameters good?

- Parameters recover input-output mapping on training data
- How can we know they do not simply memorize training data?

# Questions

Is the loss function appropriate?

~~Quadratic loss: convex cost, closed-form solution~~

But does the optimized quantity indicate classifier's performance?

Is the classifier appropriate?

Linear classifier: fast computation

But could e.g. a non-linear classifier have better performance?

Are the estimated parameters good?

Parameters recover input-output mapping on training data

How can we know they do not simply memorize training data?

# Lecture outline

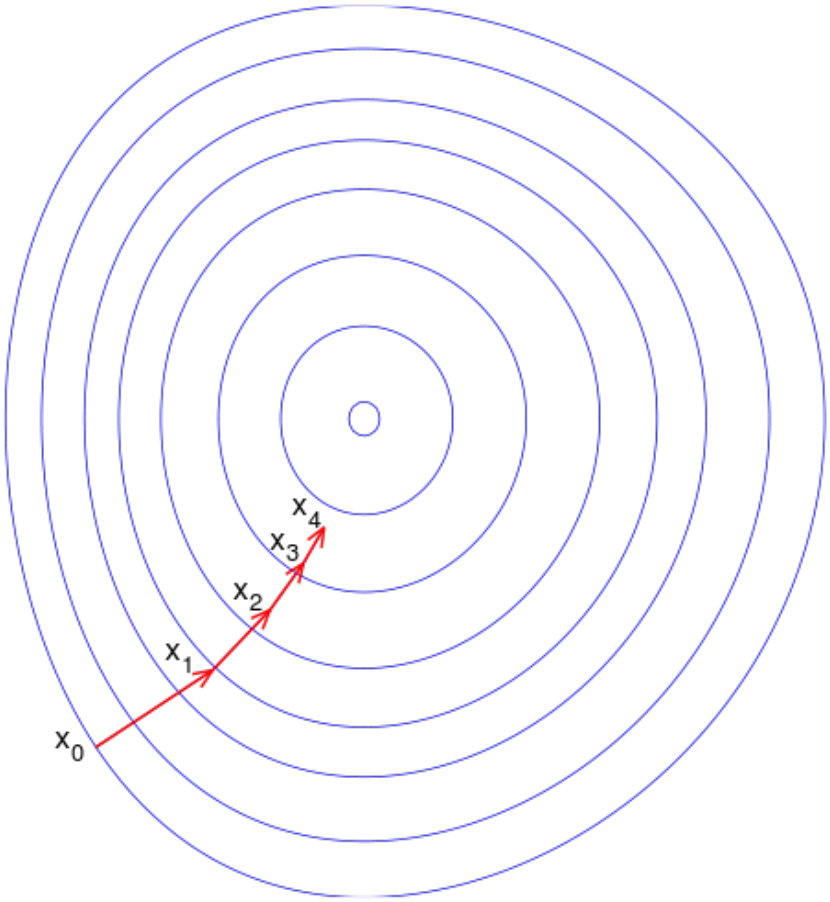
Recap & problems of linear regression

Logistic Regression

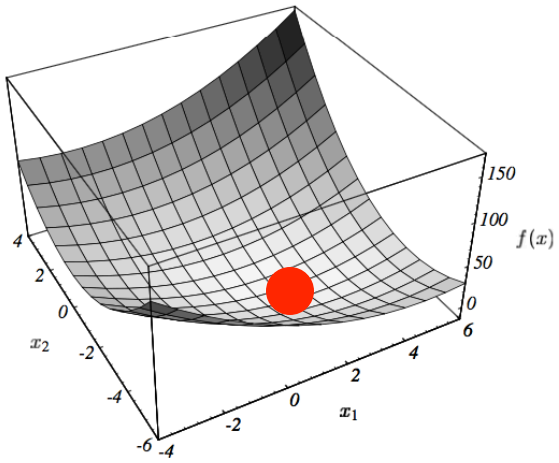
Training criterion formulation

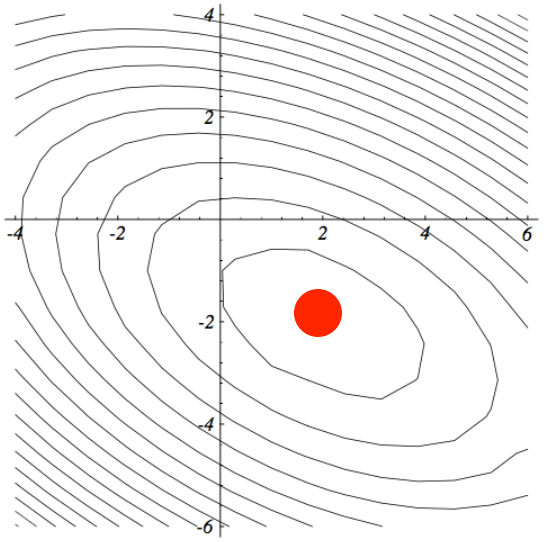
Interpretation

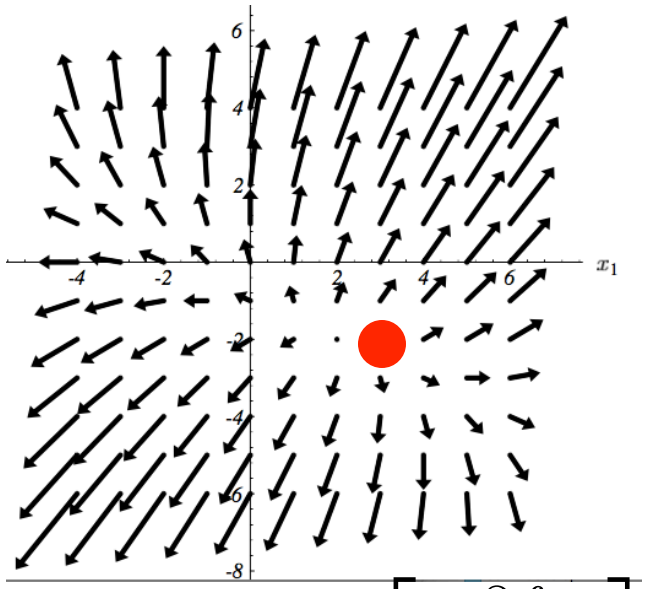
Optimization



# Gradient-based optimization




$f(\mathbf{x})$

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

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

**2D function graph**

**isocontours**

**gradient field**

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

# Gradient of cross-entropy los

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

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

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

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

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

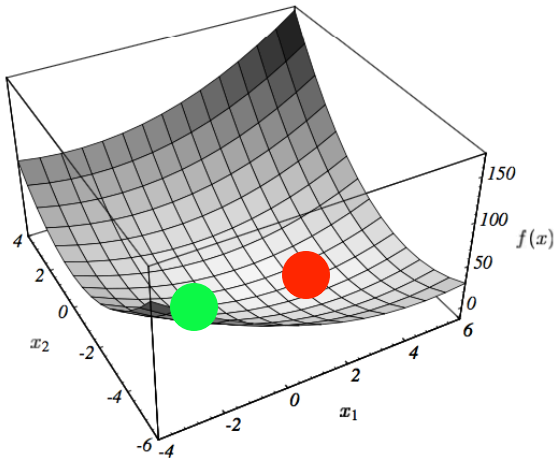
$$= -\sum_{i=1}^{N}\left[y^i - g(\mathbf{w}^T\mathbf{x}^i)\right] \mathbf{x}_k^i$$

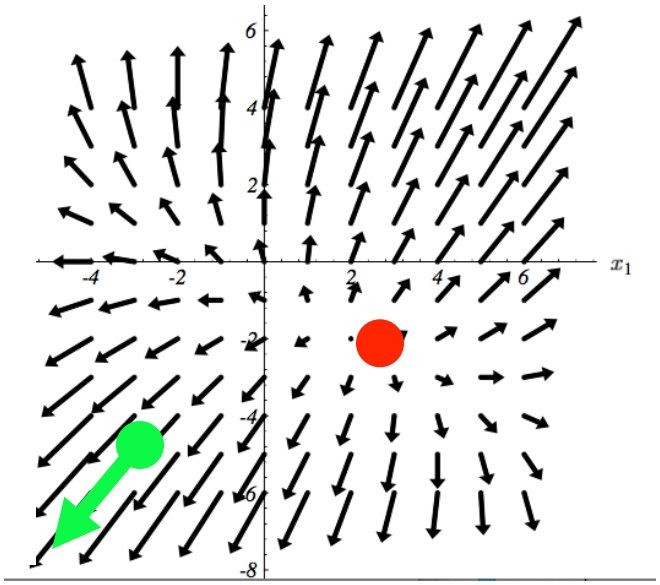
**Nonlinear system of equations!!**

**How can we find where this becomes zero?**

**Let's make it happen!**

# Gradient-based minimization

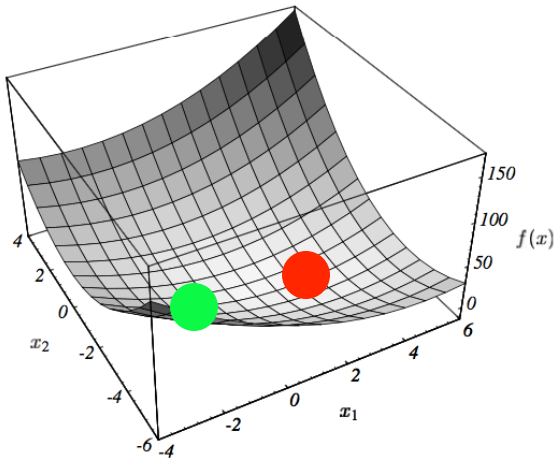


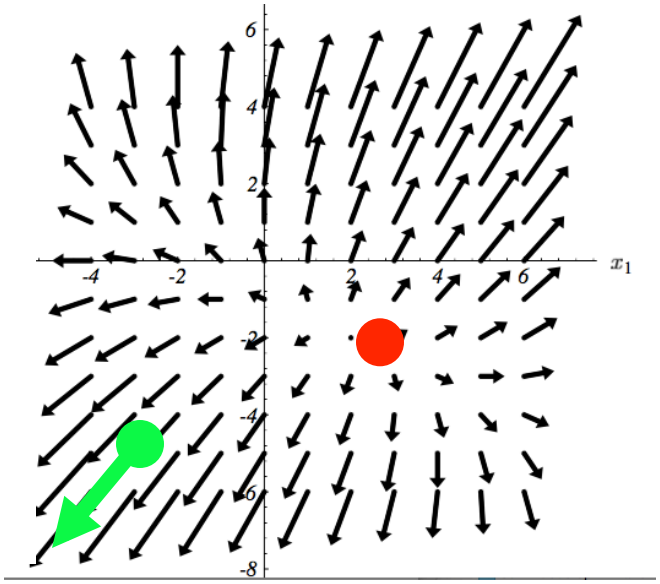




$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Fact: gradient at any point gives direction of fastest increase

# Gradient-based minimization



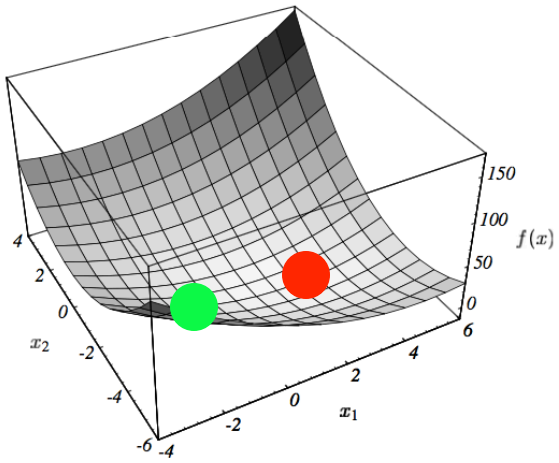



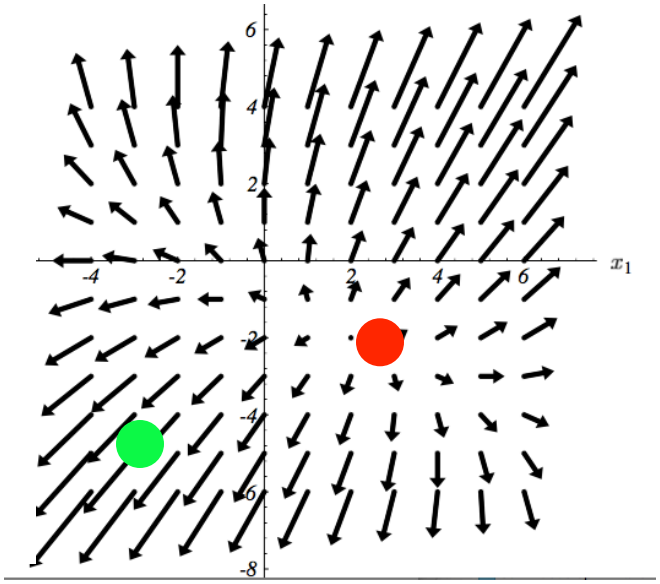

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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-based minimization



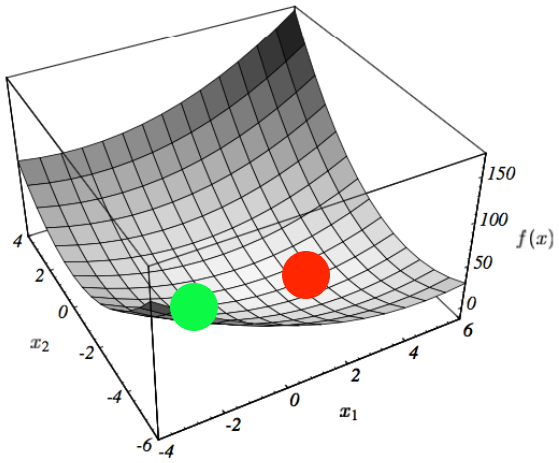



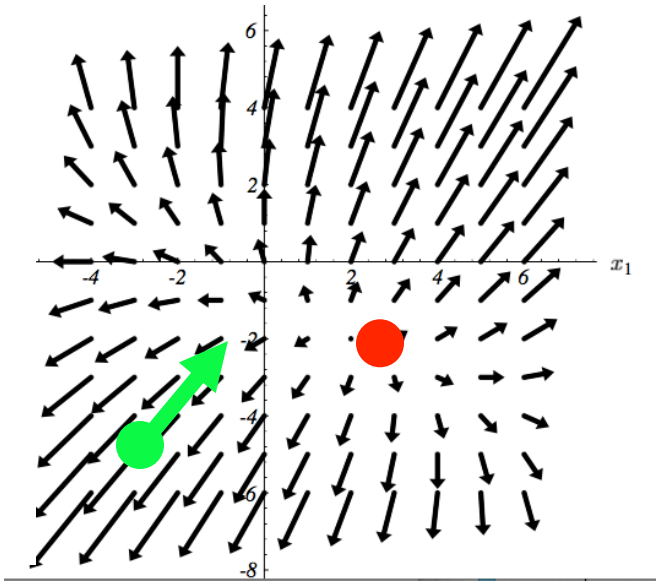

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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-based minimization



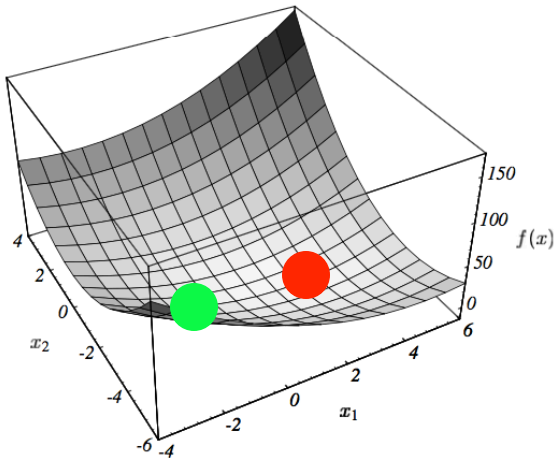



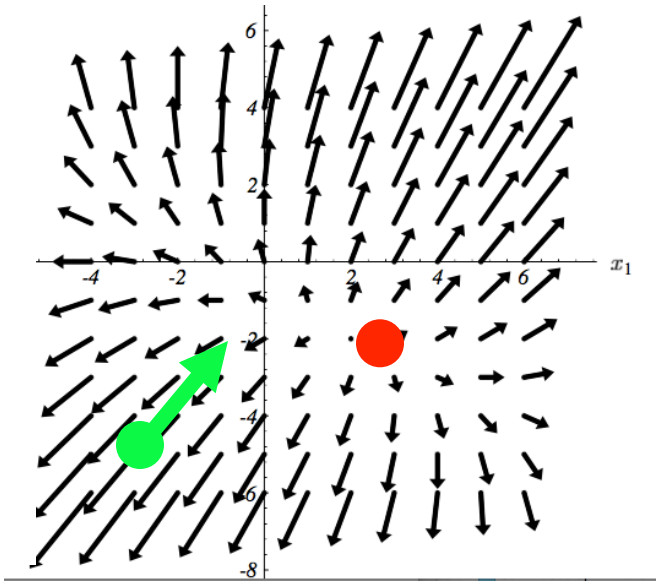

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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-based minimization




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

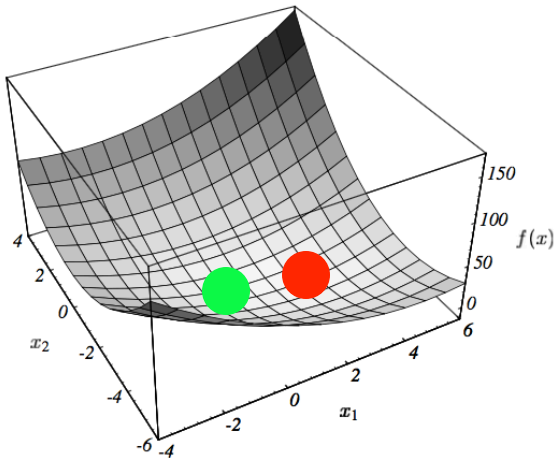
Fact: gradient at any point gives direction of fastest increase

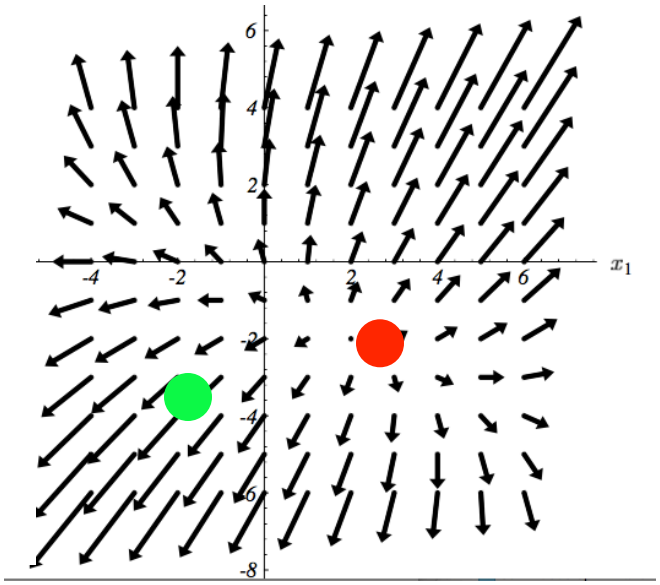
Idea: start at a point and move in the direction opposite to the gradient

Initialize: $\mathbf{x}_0$

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

# Gradient-based minimization

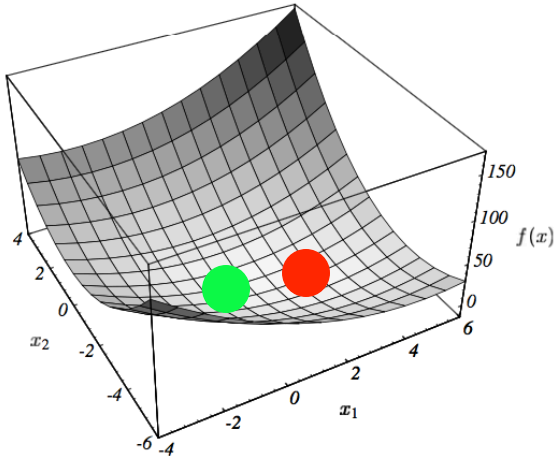


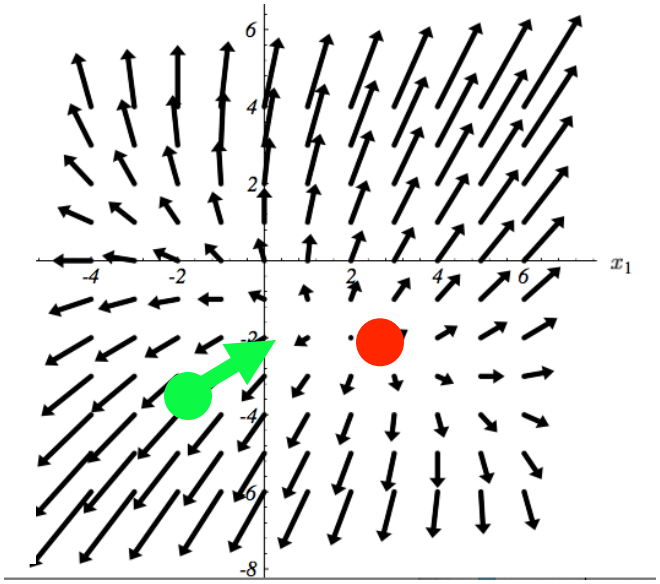




$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

# Gradient-based minimization

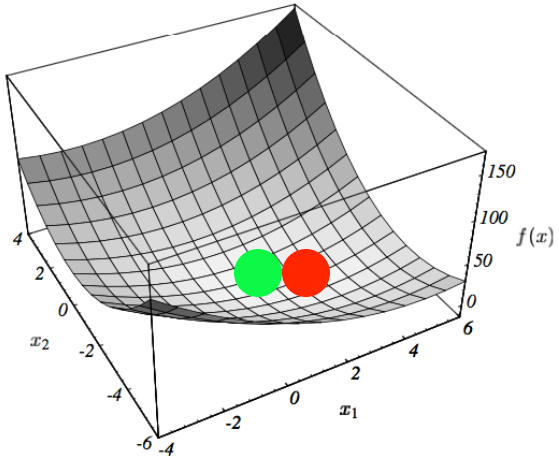


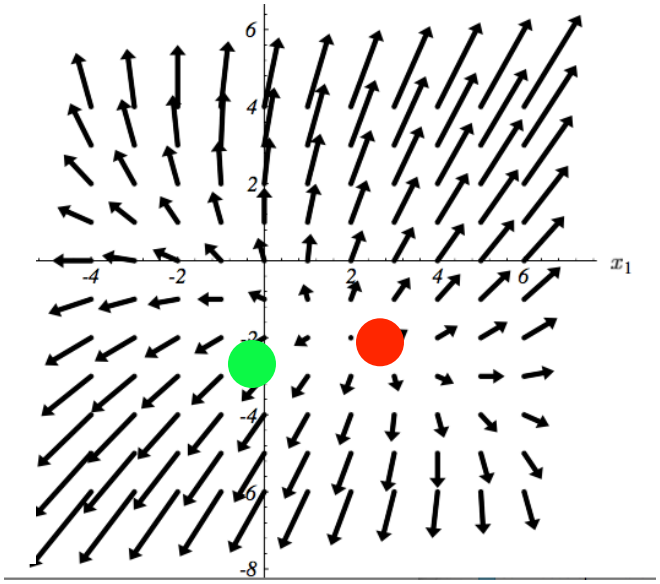




$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

# Gradient-based minimization

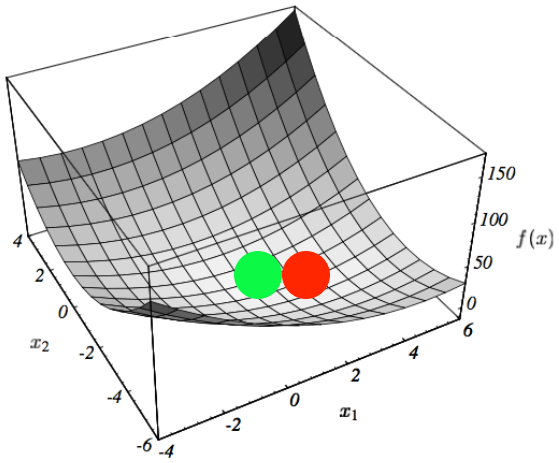


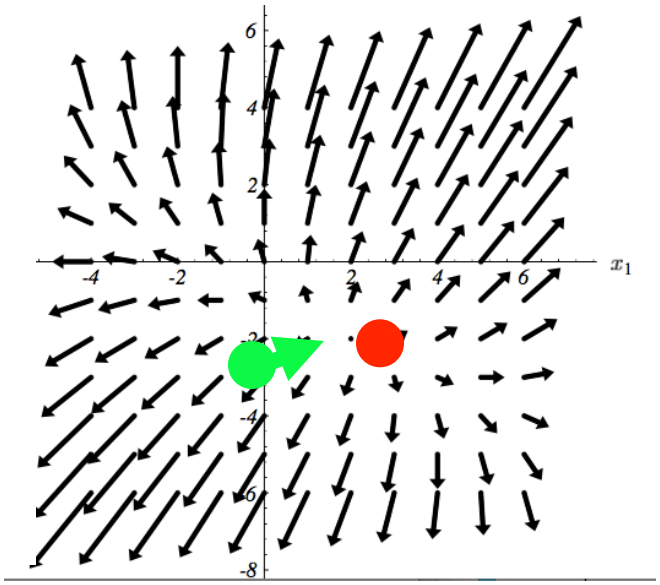




$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

# Gradient-based minimization

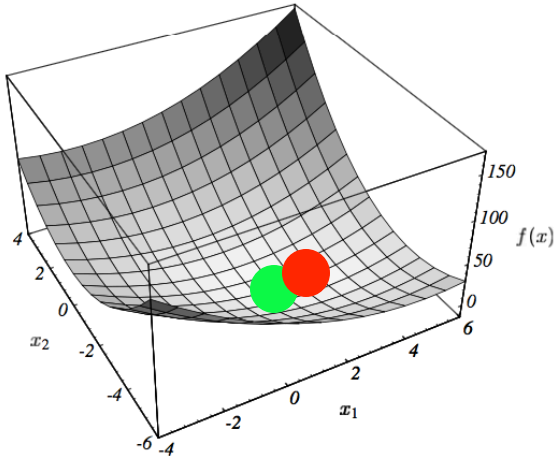


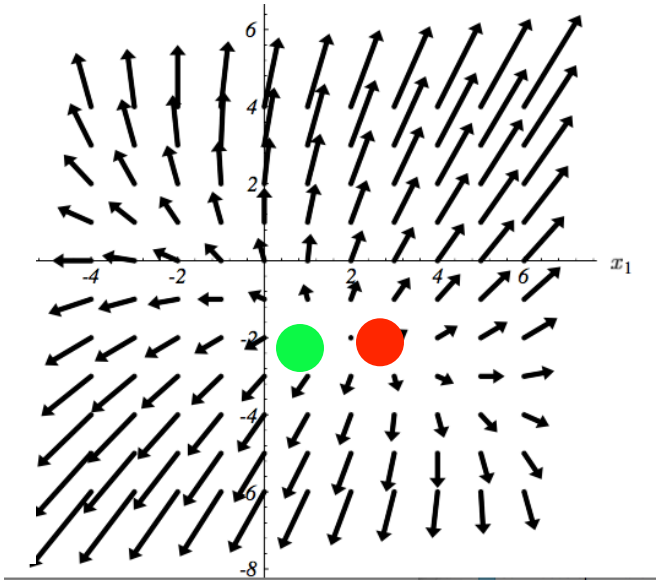




$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

# Gradient-based minimization

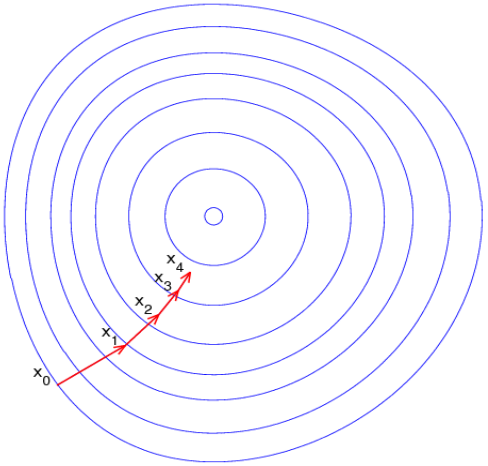



$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

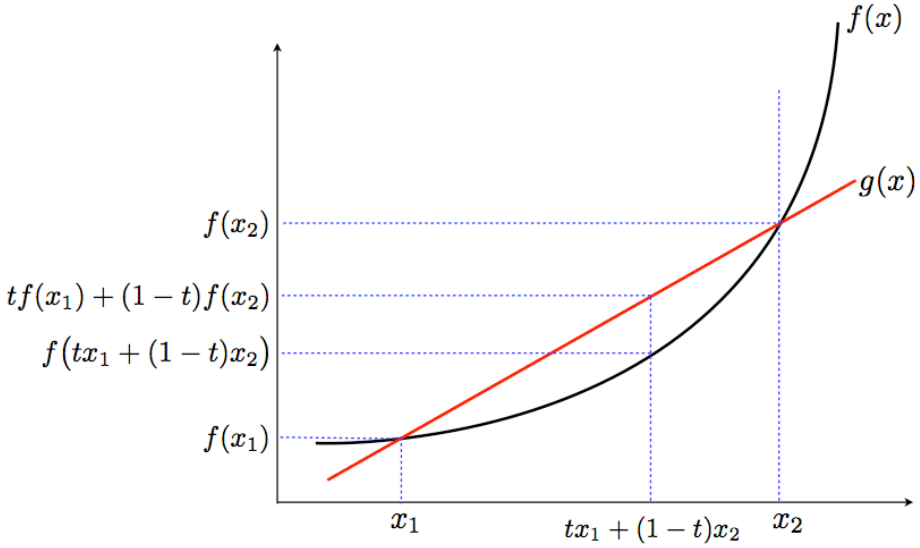
# Gradient descent minimization method

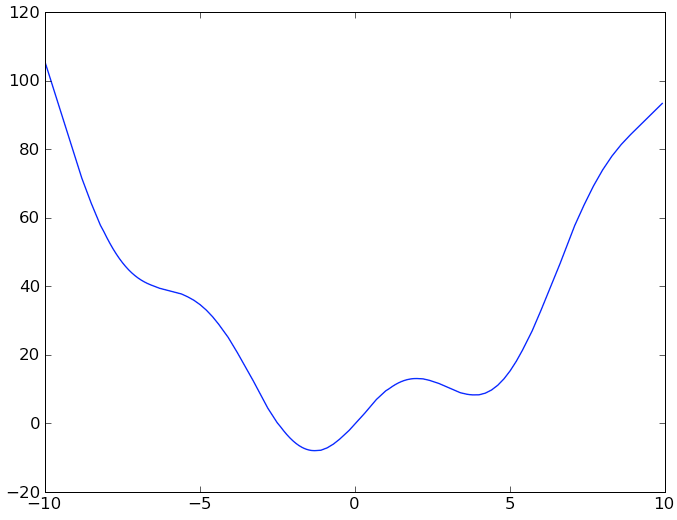



Initialize: $\mathbf{x}_0$

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

We can always make it converge for a convex function



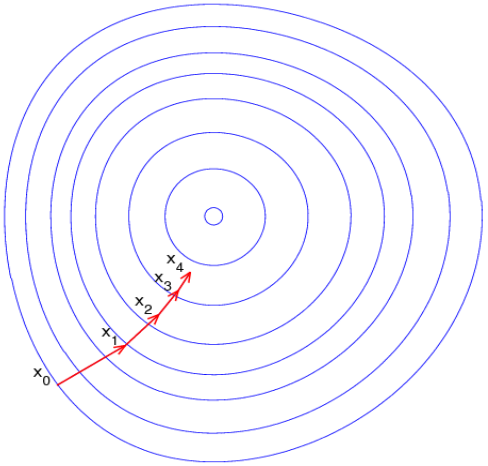



# Problems of gradient descent

**Step-size selection:**

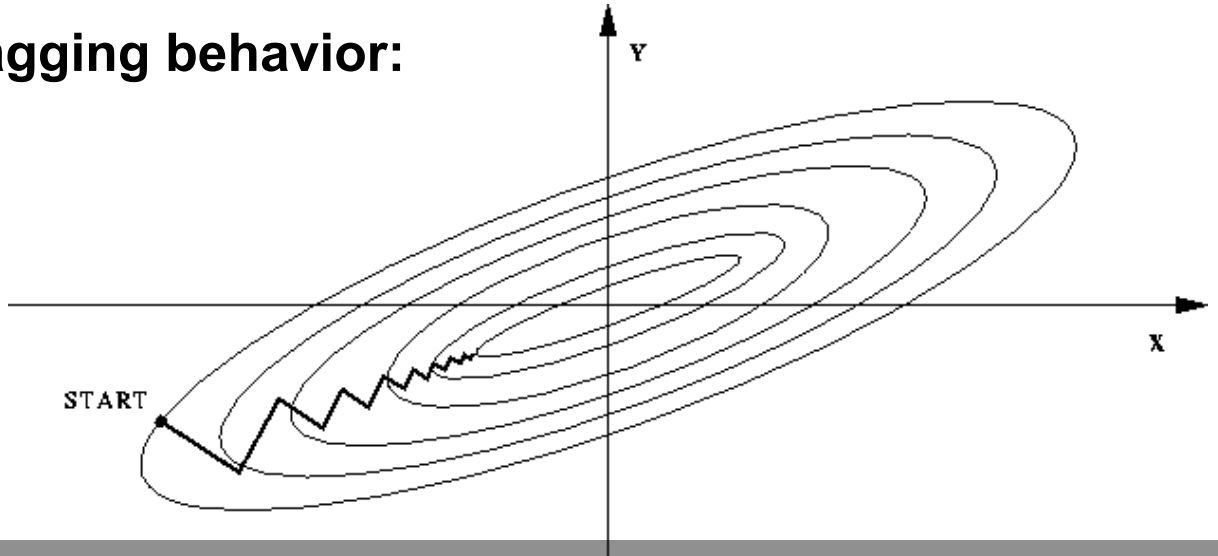
Initialize: $\mathbf{x}_0$

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




**How to set this?**

**Zig-zagging behavior:**



# Thought experiment: least squares

**Sum of squared error minimization:**

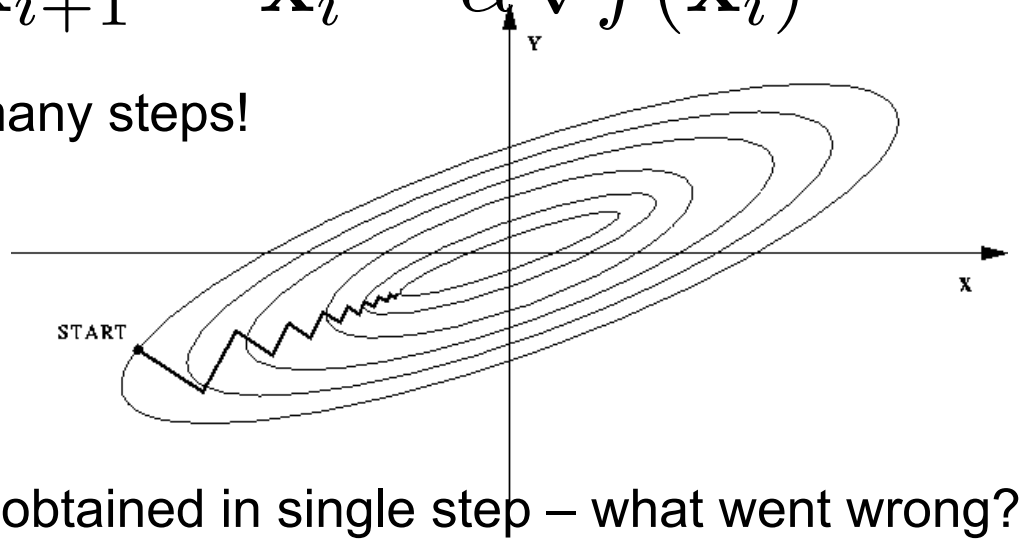
$$f(\mathbf{x}) = (\mathbf{y} - \mathbf{D}\mathbf{x})^T(\mathbf{y} - \mathbf{D}\mathbf{x})$$

**Gradient descent:**

Initialize: $\mathbf{x}_0$

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

May require many steps!




We know solution can be obtained in single step – what went wrong?

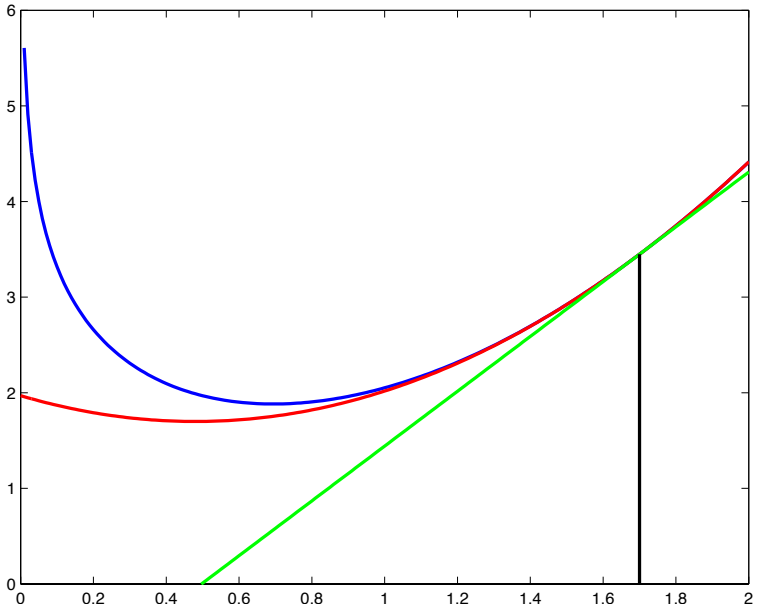
# Second-order methods

**First- order Taylor series approximation:**

$$f(x) \simeq f(a) + (x-a)f'(a) + e(x)$$

**Second-order Taylor series approximation:**

$$f(x) = f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a) + e(x)$$




**blue:**

$$f(x) = x^2 - \log(b) + \exp(\frac{x}{20})$$

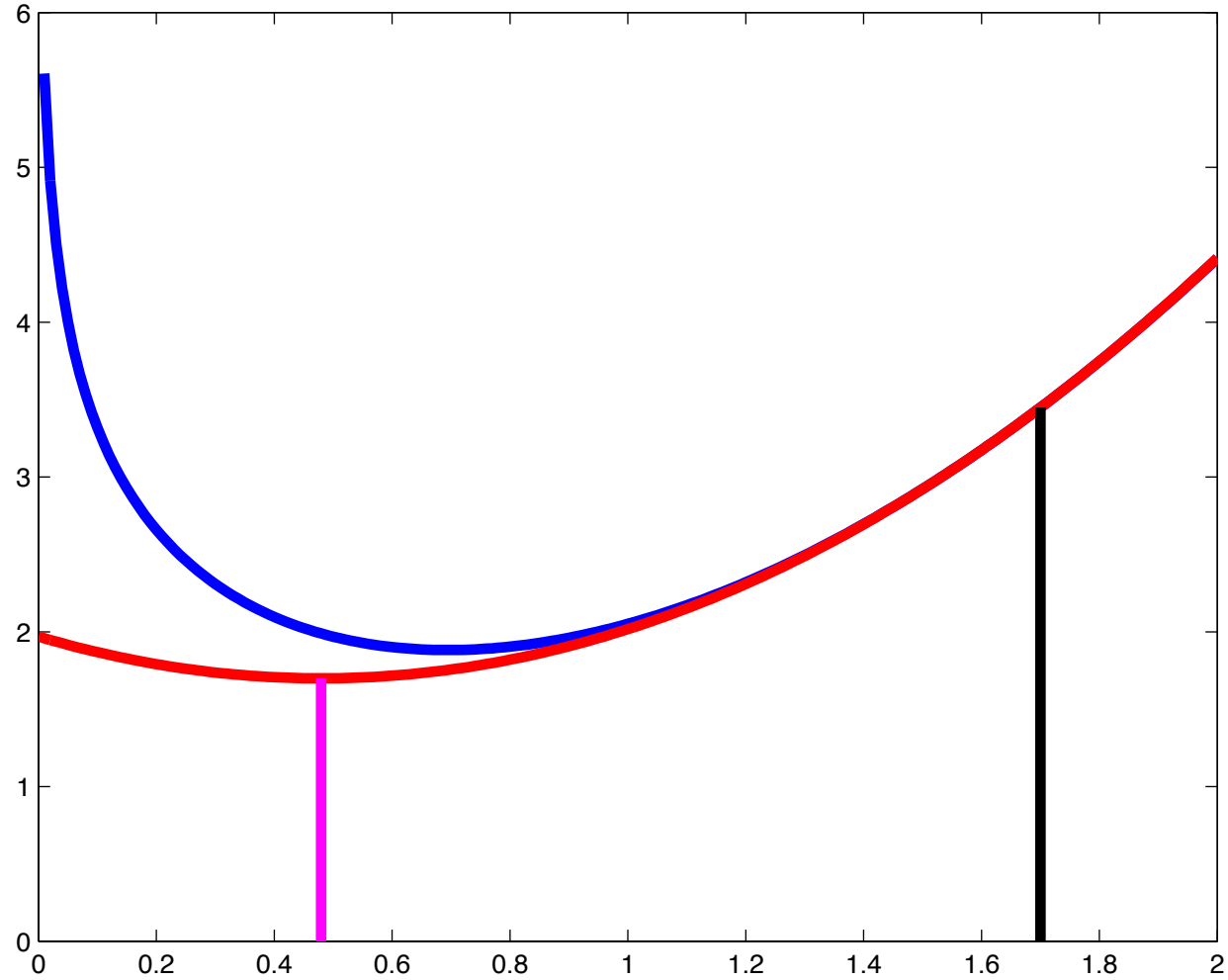
**green: linear approximation**

$$l(x) = f(a) + (x-a)f'(a)$$

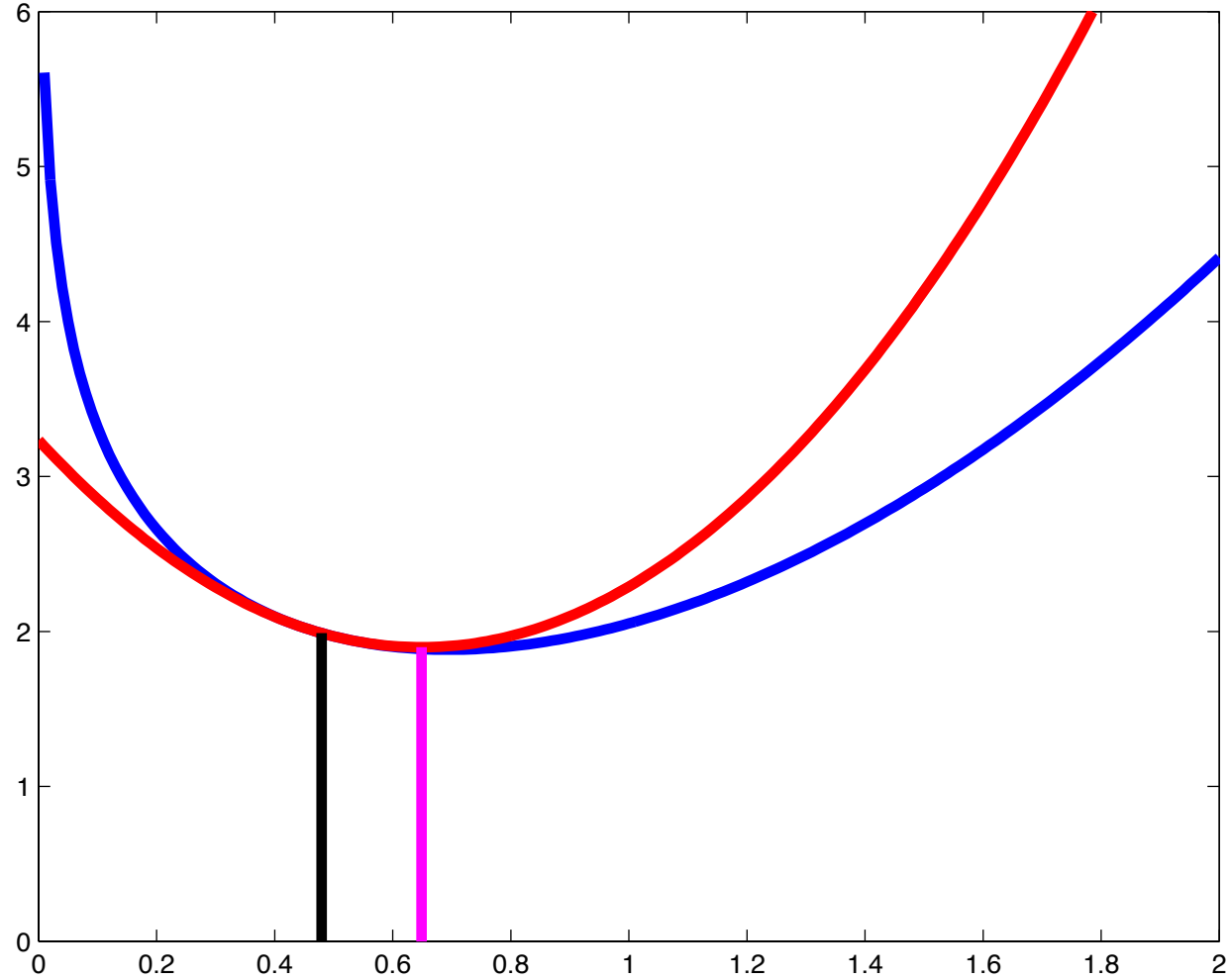
**red: quadratic approximation**

$$q(x) = f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a)$$

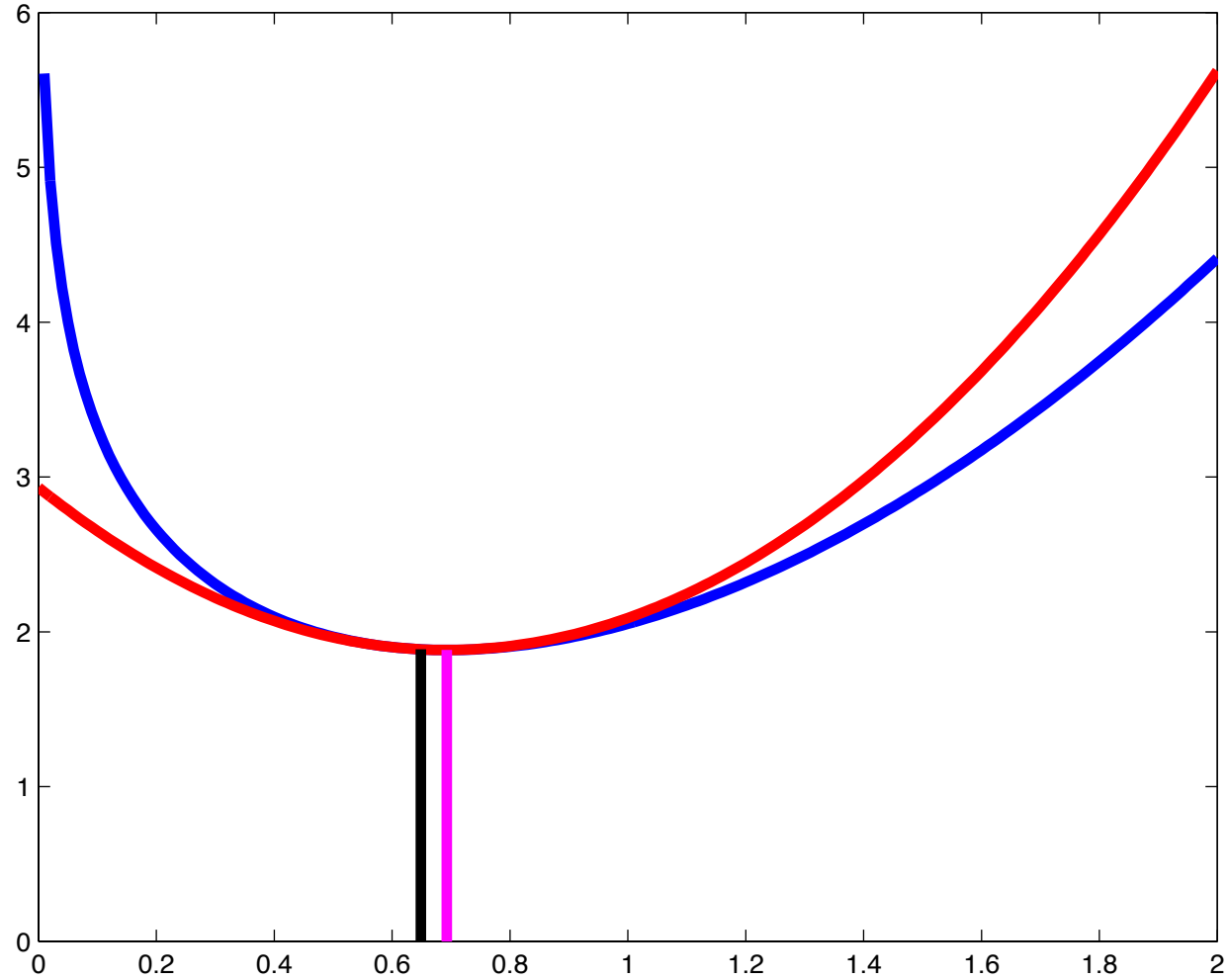
# Second-order minimization, 1D



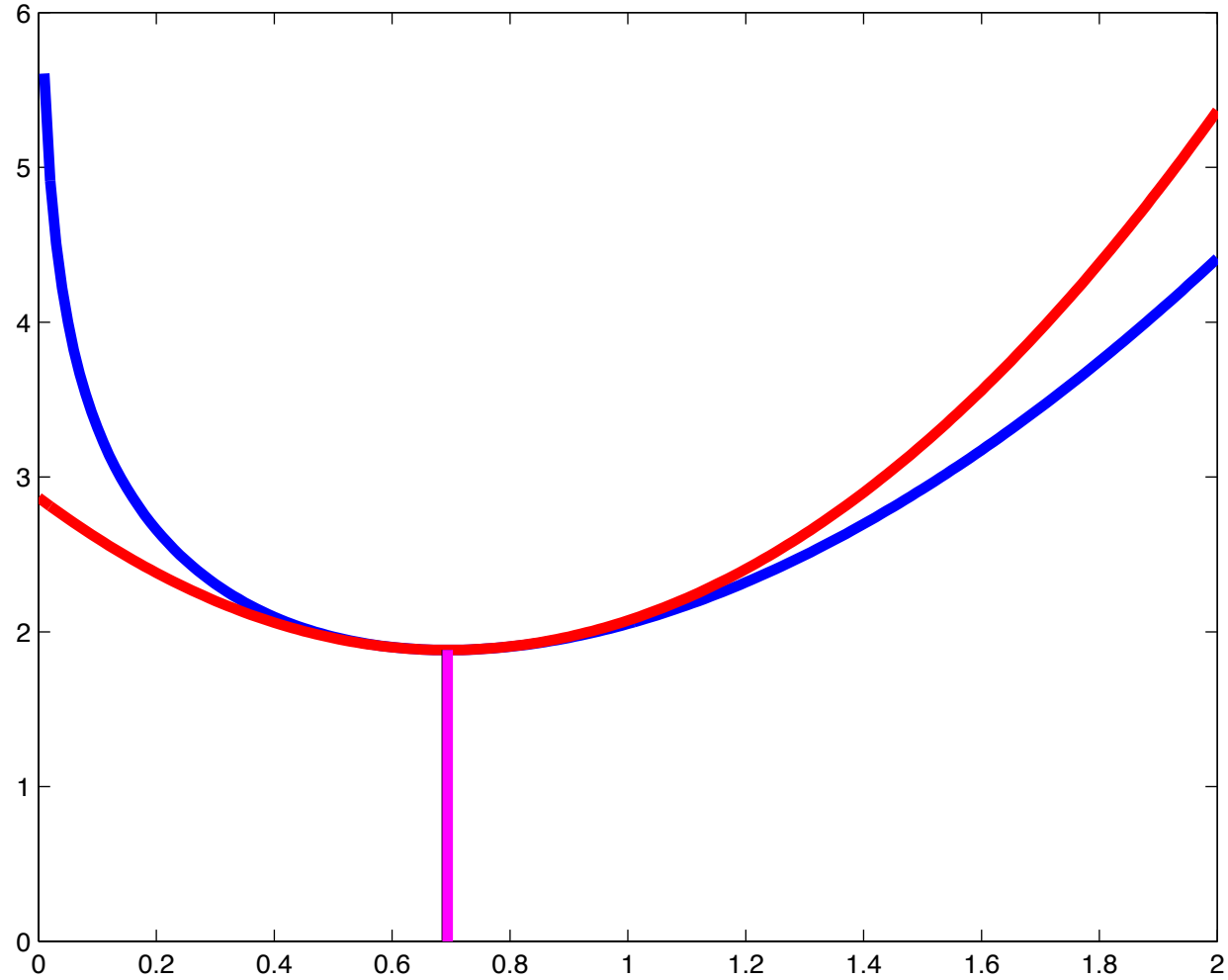
# Second-order minimization, 1D



# Second-order minimization, 1D



# Second-order minimization, 1D



# Second-order minimization, 1D

Start from some inititial position, $x_0$

At any point, form quadratic approximation:

$$f(x) \simeq q(x) = f(x_i) + (x - x_i) f'(x_i) + \frac{1}{2}(x - x_i)^2 f''(x_i)$$

Condition for minimum of quadratic approximation:

$$q'(x) = 0 \rightarrow f'(x_i) + (x - x_i) f''(x_i) = 0$$

Set point in next iteration to be at the minimum of present approximation

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

Until update is too small

# Second-order methods, multivariate case

**First- order Taylor series approximation:**

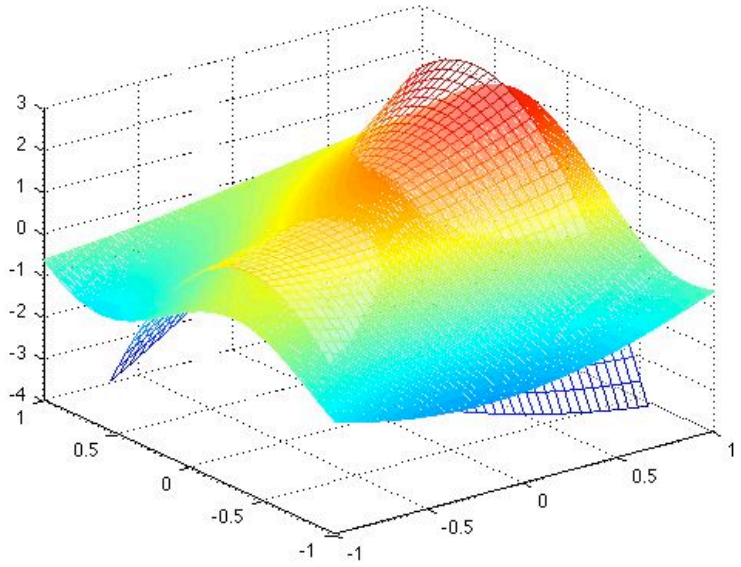
$$f(\mathbf{x}) \simeq f(\mathbf{x}_i) + (\mathbf{x} - \mathbf{x}_i)^T \nabla f(\mathbf{x}_i)$$

**Second-order Taylor series approximation:**

$$f(\mathbf{x}) \simeq f(\mathbf{x}_i) + (\mathbf{x} - \mathbf{x}_i)^T \nabla f(\mathbf{x}_i) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^T \mathbf{H} (\mathbf{x} - \mathbf{x}_i)$$

$$\doteq q(\mathbf{x})$$

$$H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$



# Second-order minimization, 1D

Start from some inititial position, $x_0$

At any point, form quadratic approximation:

$$f(x) \simeq q(x) = f(x_i) + (x - x_i) f'(x_i) + \frac{1}{2}(x - x_i)^2 f''(x_i)$$

Condition for minimum of quadratic approximation:

$$q'(x) = 0 \rightarrow f'(x_i) + (x - x_i) f''(x_i) = 0$$

Set point in next iteration to be at the minimum of present approximation

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

Until update is too small

# Second-order minimization, N-D (Newton-Raphson)

Start from some inititial position, $\mathbf{x}_0$

At any point, form quadratic approximation:

$$f(\mathbf{x}) \simeq q(\mathbf{x}) = f(\mathbf{x}_i) + (\mathbf{x} - \mathbf{x}_i)^T \nabla f(\mathbf{x}_i) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^T \mathbf{H}(\mathbf{x}_i)(\mathbf{x} - \mathbf{x}_i)$$

Condition for minimum of quadratic approximation:

$$\nabla q(\mathbf{x}) = 0 \rightarrow \nabla f(\mathbf{x}_i) + (\mathbf{x} - \mathbf{x}_i)^T \mathbf{H}(\mathbf{x}_i) = 0$$

Set point in next iteration to be at the minimum of present approximation

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

Until update is too small

# Newton-Raphson for Logistic Regression

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

Hessian:

$$\frac{\partial^2 L(\mathbf{w})}{\partial w_k \partial w_j} = \frac{\partial \left(-\sum_{i=1}^{N} \left[y^i - g(\mathbf{w}^T\mathbf{x}^i)\right] \mathbf{x}_k^i\right)}{\partial w_j}$$

$$= \sum_{i=1}^{N} \mathbf{x}_k^i \frac{\partial g(\mathbf{w}^T\mathbf{x}^i)}{\partial w_j} = \sum_{i=1}^{N} \mathbf{x}_k^i g(\mathbf{w}^T\mathbf{x}^i)(1 - g(\mathbf{w}^T\mathbf{x}^i))\mathbf{x}_j^i$$

$$H(\mathbf{w}) = \mathbf{X}^T\mathbf{R}\mathbf{X}, \quad R_{i,i} = g(\mathbf{w}^T\mathbf{x}^i)(1 - g(\mathbf{w}^T\mathbf{x}^i))$$

# Multiple classes & linear regression

K classes: one-of-k coding

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

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix} (N \times K) \qquad \mathbf{X} = \begin{bmatrix} x^1 \\ \vdots \\ x^N \end{bmatrix} (N \times M)$$

$$\mathbf{W} = \left[\mathbf{w}^1 | \mathbf{w}^2 | \ldots | \mathbf{w}^K\right] (M \times K)$$

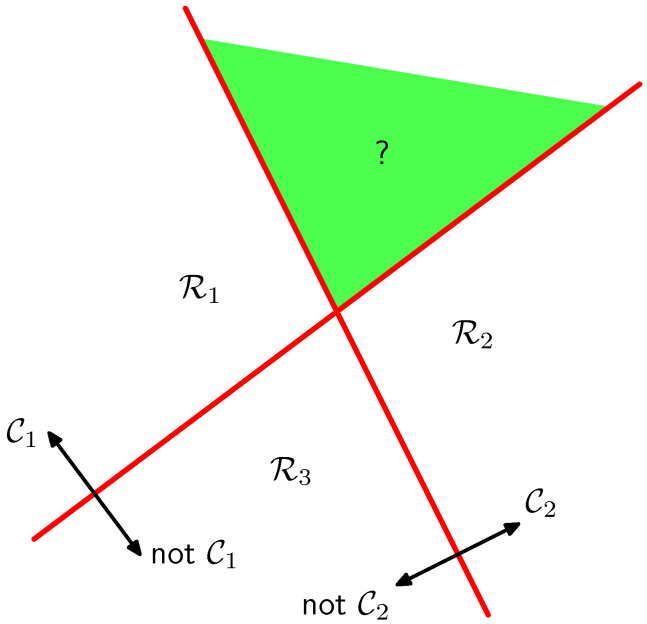
Loss function: $$RSS(\mathbf{W}) = \mathrm{Trace}\left[(\mathbf{Y} - \mathbf{XW})^T (\mathbf{Y} - \mathbf{XW})\right]$$

Least squares fit: $$\hat{\mathbf{W}} = (\mathbf{X^T X})^{-1}(\mathbf{X^T Y})$$

# Multiple classes & linear regression

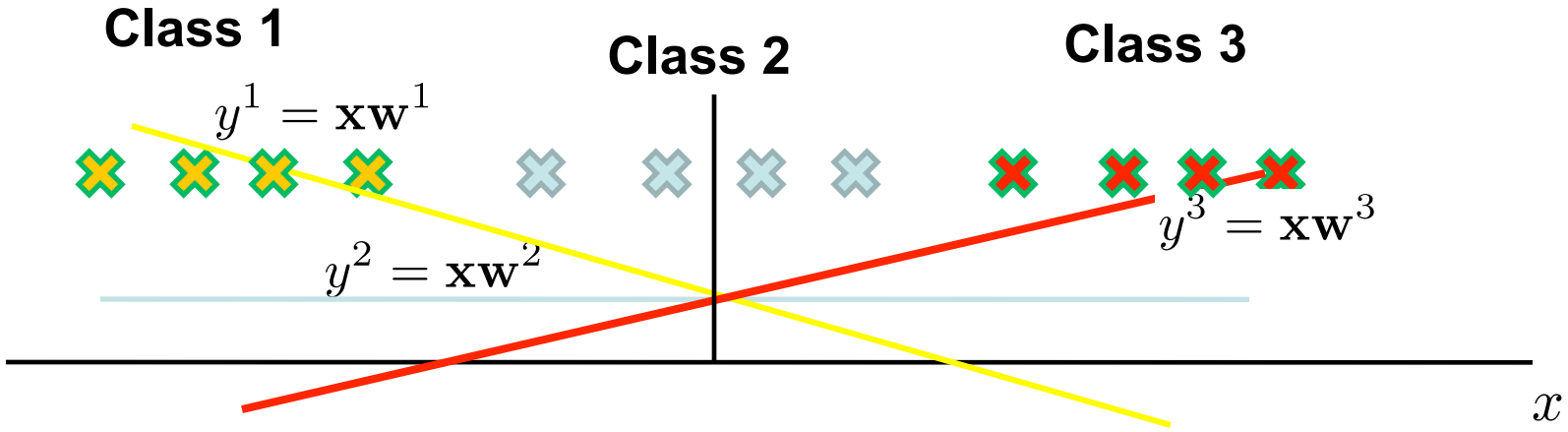
One linear discriminant per class: $y^k = \sum_{i=1}^{M} x_i w_i^k = \mathbf{x}\mathbf{w}^k$

Problem: ambiguous regions



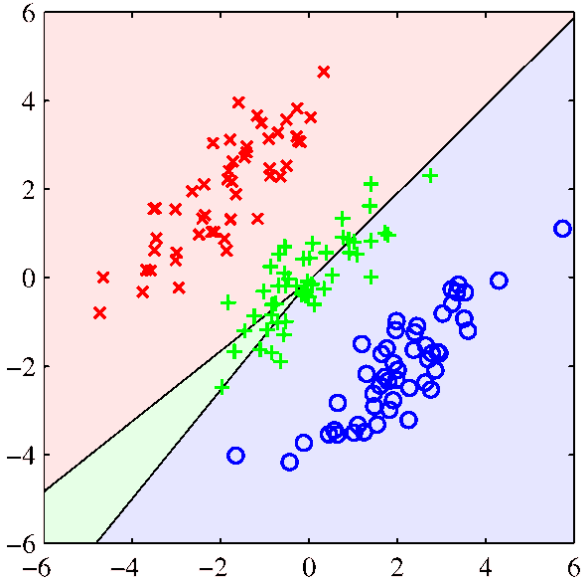

Solution: assign to discriminant with largest score

# Masking Problem in linear regression




Nothing ever gets assigned to class 2!

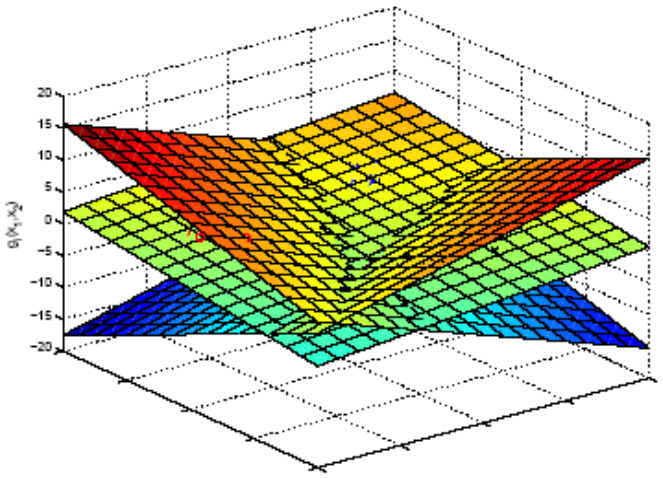
2D version:



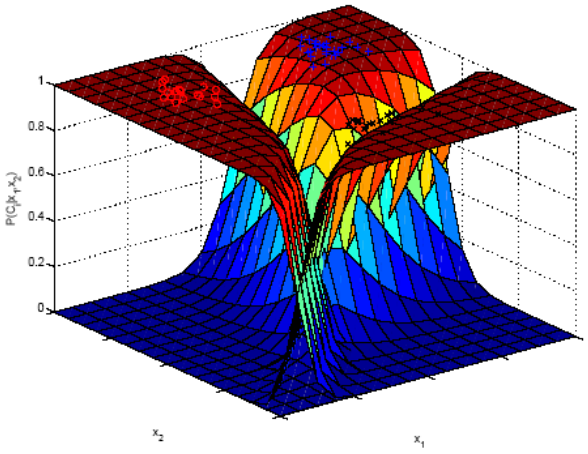
# Multiple classes & logistic regression

Soft maximum of competing classes:

$$P(y = k|\mathbf{x}) \quad = \quad \frac{\exp(\mathbf{x}\mathbf{w_k})}{\sum_{n=1}^{K} \exp(\mathbf{x}\mathbf{w_n})},$$



**Discriminants (inputs)**



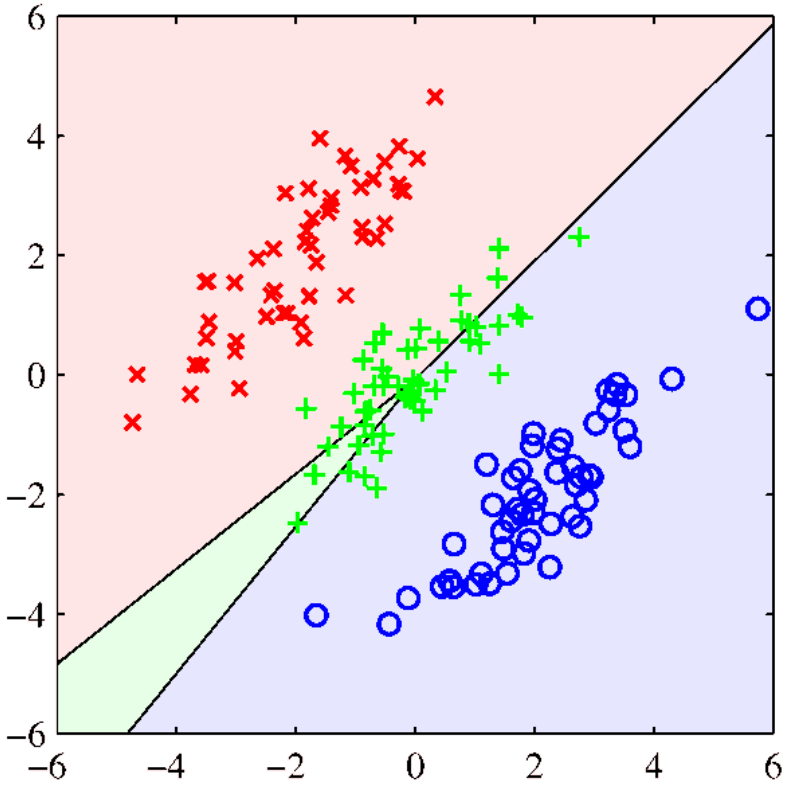
**Softmax (outputs)**

Label probability: $P(y^i|\mathbf{x}^i, \mathbf{w}) = \prod_{k=1}^{K} g_k(\mathbf{x}\mathbf{w})^{[y^i=k]}$

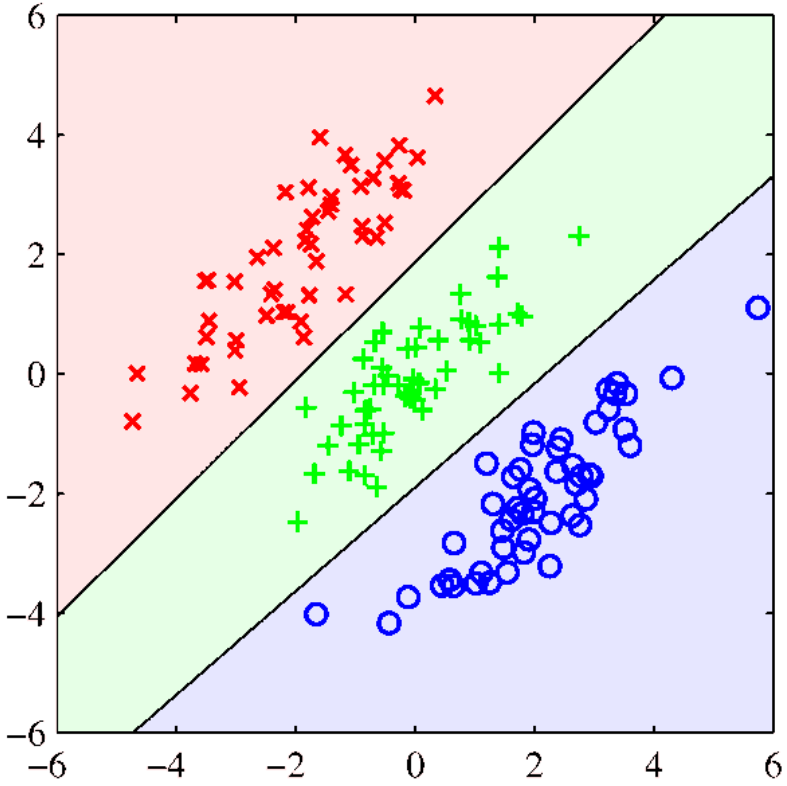
Similar steps for parameter estimation

# Logistic vs Linear Regression, n>2 classes

Linear regression




Logistic regression




Logistic regression does not exhibit the masking problem