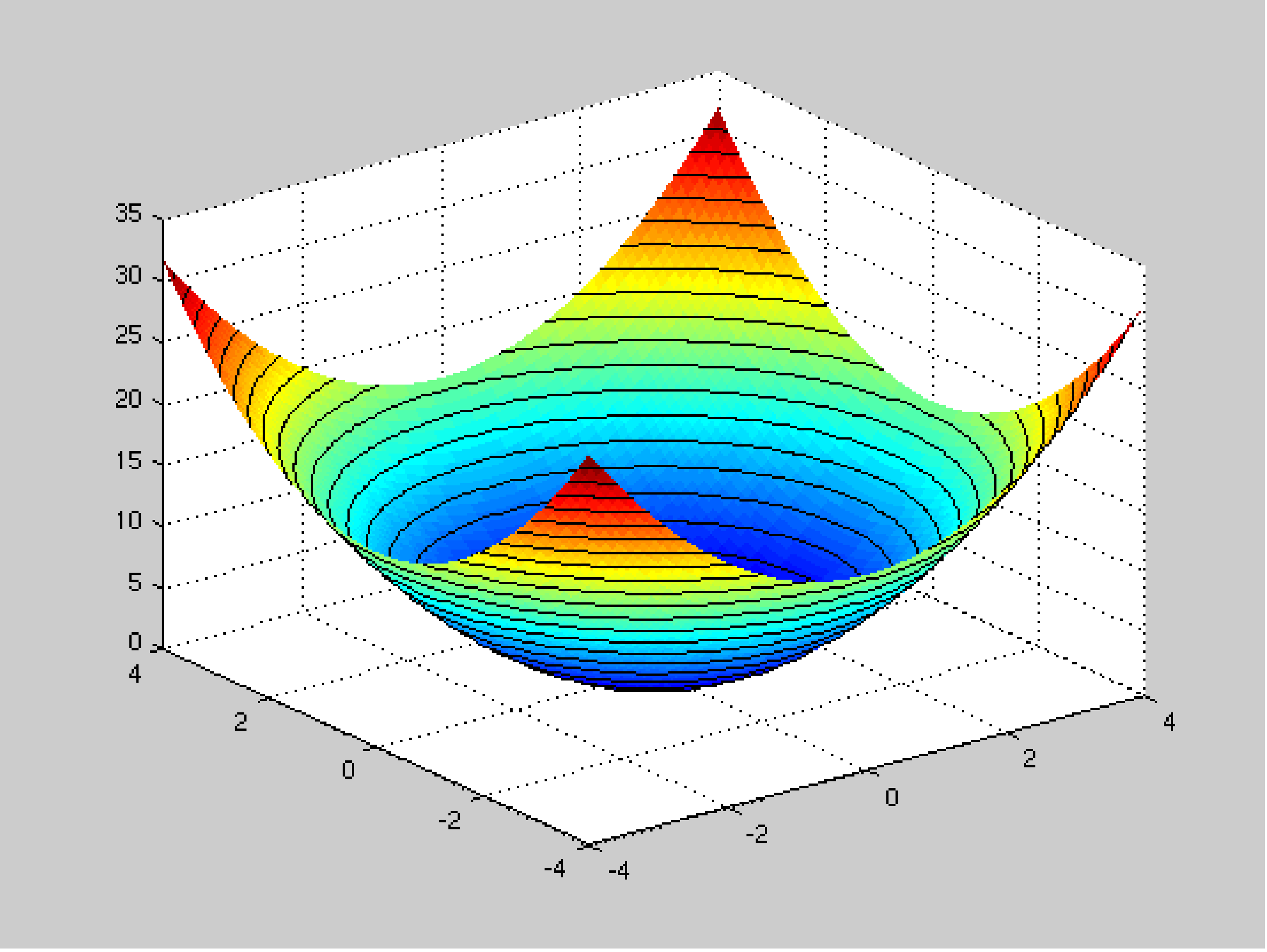
CSE463: Neural Networks
Optimization for Engineers

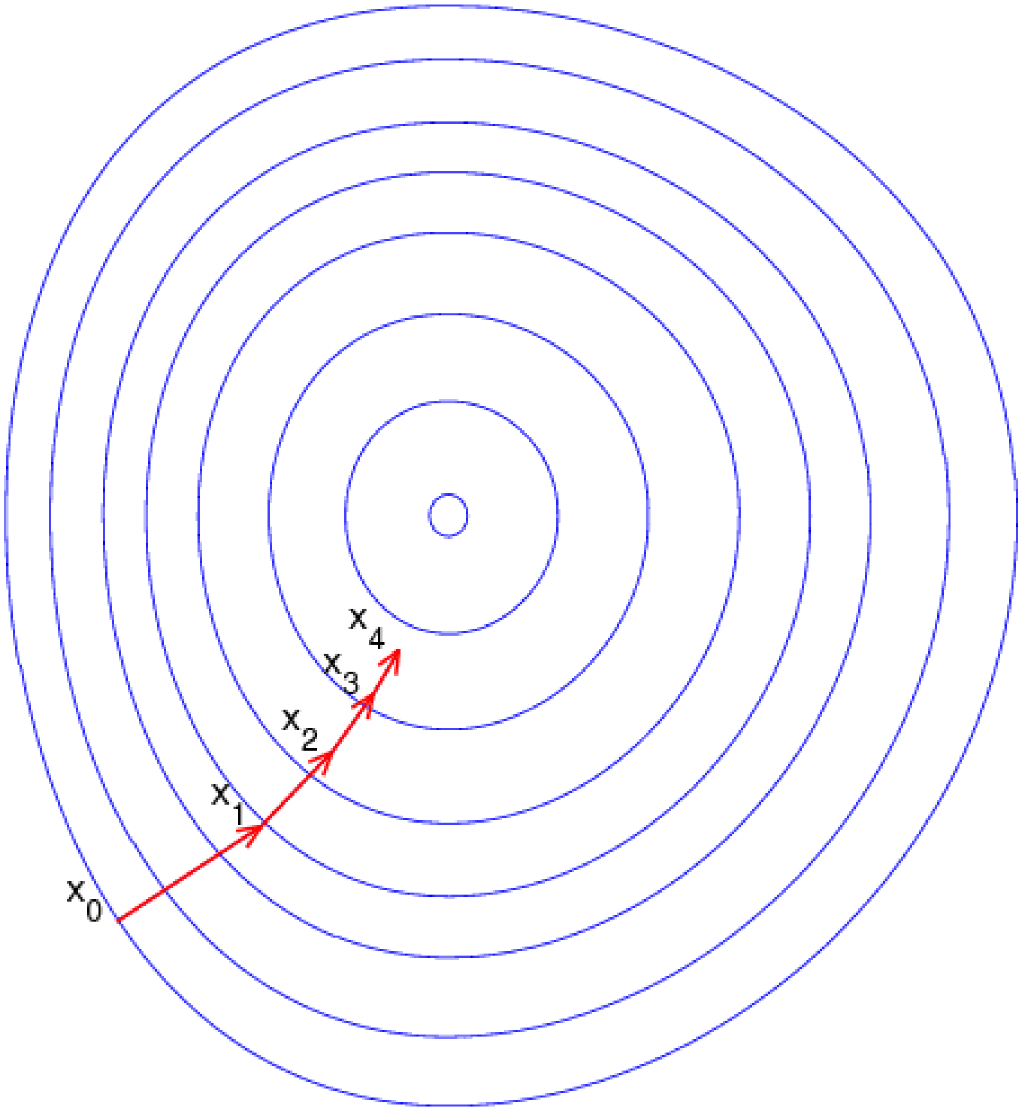
by:
Hossam Abd El Munim
Computer & Systems Engineering Dept.,
Ain Shams University,
1 El-Sarayat Street, Abbassia, Cairo 11517

Gradient Descent Optimization

Assume the following 2D Function



Contour plot



Gradient descent: head downhill

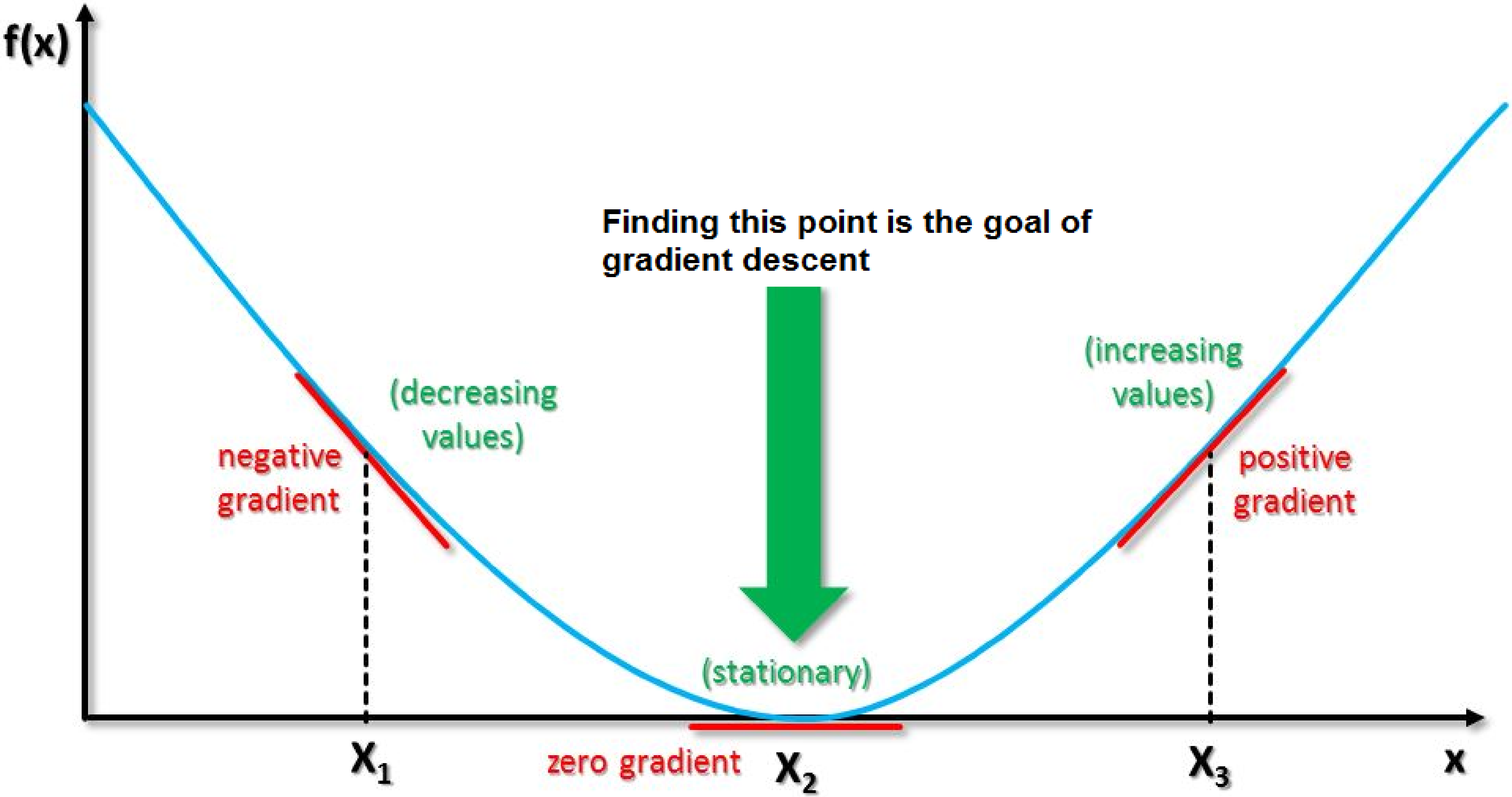
http://en.wikipedia.org/wiki/Gradient_descent

The Gradient Descent

**We select moving in the gradient direction such that:-

$$f(X_0) \geq f(X_1) \geq f(X_2) \ldots \geq f(X_{n-1}) \geq f(X_n)$$

The Gradient Descent Algorithm



The Gradient Descent Algorithm

Step 0: $Select\ X_0 \in R^n,\ Set\ \alpha, and\ i = 0$

Step 1: Compute $\nabla f(X_i)$

Step 2: if $||\nabla f(X_i)|| < \varepsilon, Stop$
Otherwise Go To Step 3

Step 3: Compute $X_{i+1} = X_i - \alpha \nabla f(X_i)$

Step 4: Update i=i+1

Step 5: Go To Step 1

Computing the Gradient

$$f : R^n \to R$$

$$\nabla f(x_1,\ldots,x_n) := \left(\frac{\partial f}{\partial x_1},\ldots, \quad \frac{\partial f}{\partial x_n} \right)$$

Step Size Selection (α)

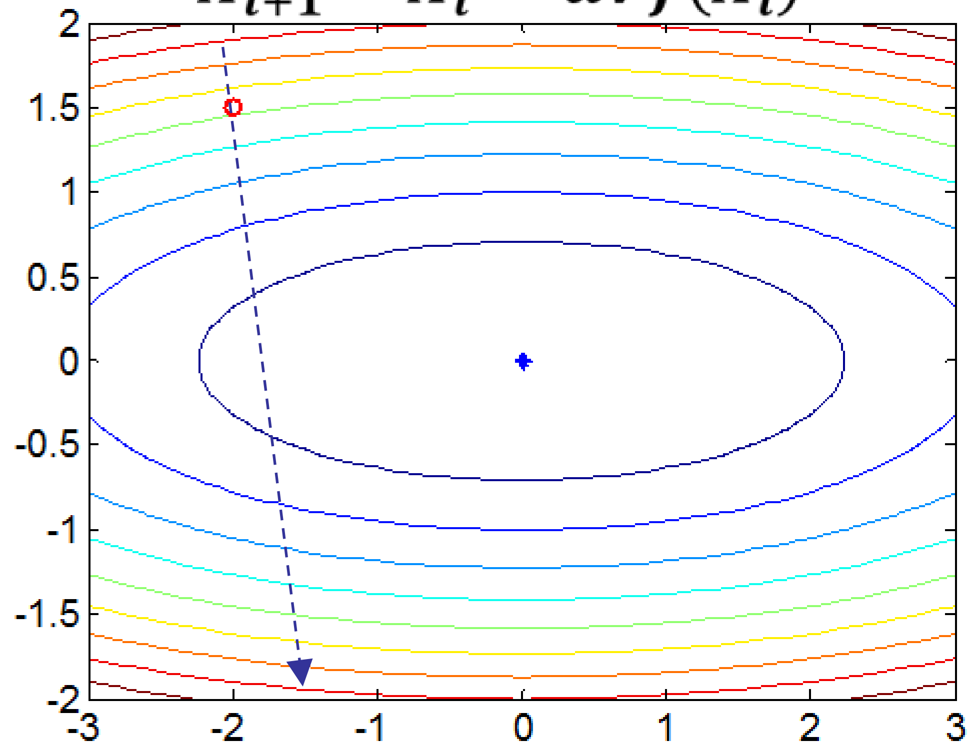
How should we select the step size?

- α too small: convergence takes long time
- α too large: overshoot minimum

Line minimization:

$$\alpha = \arg\min_{\alpha} f(X_i - \alpha \nabla f(X_i))$$

$$X_{i+1} = X_i - \alpha \nabla f(X_i)$$



The Steepest Gradient Descent Algorithm (Line Search)

Step 0: $Select\ X_0 \in R^n,\ Set\ i = 0$

Step 1: Compute $\nabla f(X_i)$

Step 2: if $||\nabla f(X_i)|| < \varepsilon, Stop$
Otherwise Go To Step 3

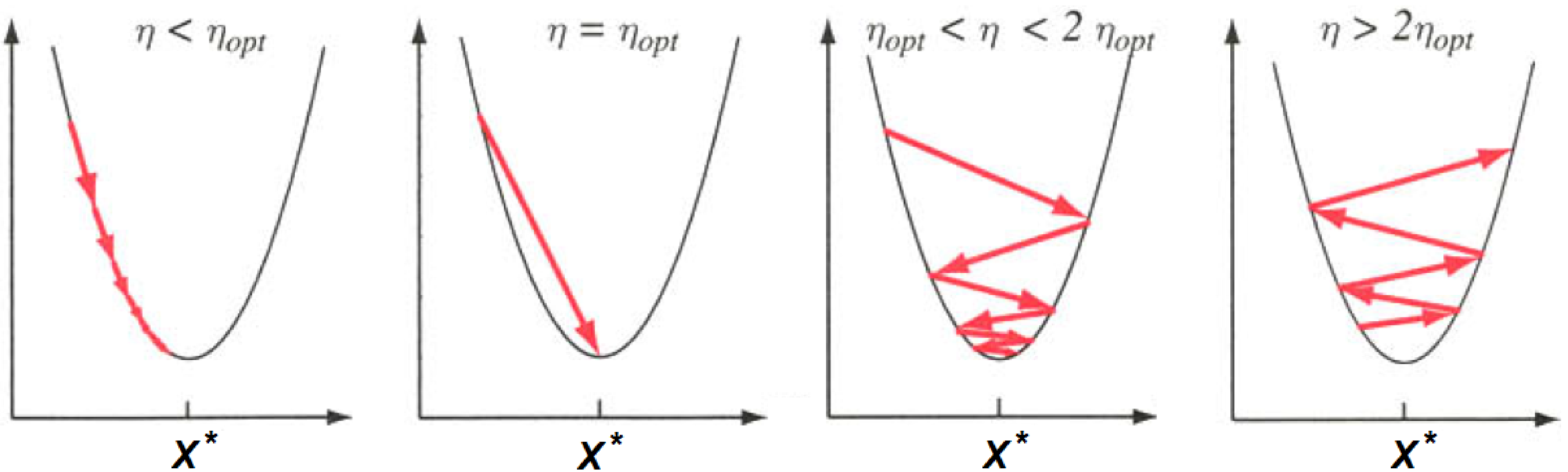
Step 3: Update $\alpha^* = \underset{\alpha}{\operatorname{argmin}} f(X_i - \alpha \nabla f(X_i))$

Step 4: Compute $X_{i+1} = X_i - \alpha^* \nabla f(X_i)$

Step 5: Update i=i+1

Step 6: Go To Step 1

The Steepest Gradient Descent Algorithm



Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges.

Step Size Automatic Selection: The Newton-Raphson Algorithm

Step 0: $Select\ X_0 \in R^n,\ \ and\ i = 0$

Step 1: Compute $\nabla f(X_i)\ and\ H$

Step 2: if $||\nabla f(X_i)|| < \varepsilon, Stop$
Otherwise Go To Step 3

Step 3: Compute $\alpha = \ H^{-1}$

Step 4: Compute $X_{i+1} = X_i\ - \alpha \nabla f(X_i)$

Step 5: Update i=i+1

Step 6: Go To Step 1

Ex1: Gradient Descent

$\alpha = 0.1$

$f(x)=x^4 - x^3 + x^2 - x + 1$ $df/dx=4x^3 - 3x^3 + 2x - 1$

	Xn	f(Xn)	df/dx
1	1	1	2
2	0.8000	0.7376	0.7280
3	0.7272	0.6967	0.4062
4	0.6866	0.6834	0.2536
5	0.6612	0.6781	0.1672
6	0.6445	0.6757	0.1137
7	0.6331	0.6746	0.0789
8	0.6252	0.6741	0.0554
9	0.6197	0.6738	0.0393
10	0.6158	0.6737	0.0280
11	0.6130	0.6736	0.0200
12	0.6110	0.6736	0.0144
13	0.6095	0.6736	0.0103
14	0.6085	0.6736	0.0074
15	0.6078	0.6736	0.0054
16	0.6072	0.6736	0.0039
17	0.6068	0.6736	0.0028
18	0.6066	0.6736	0.0020
19	0.6064	0.6736	0.0015
20	0.6062	0.6736	0.0011
21	0.6061	0.6736	7.6266e-04

Ex2: Newton Raphson

$f(x)=x^4 - x^3 + x^2 - x + 1$

$df/dx = 4x^3 - 3x^3 + 2x - 1$

$d^2f/dx^2 = 12x^2 - 9x^2 + 2$

X	f(X)	df/dx	d2f/dx2
10	9091	3719	1142
6.7434	1.8010e+03	1.1027e+03	507.2260
4.5695	357.8926	327.1530	225.1489
3.1165	71.6578	97.1690	99.8496
2.1433	14.7075	28.8891	44.2657
1.4907	3.3569	8.5650	19.7216
1.0564	1.1260	2.4805	9.0532
0.7824	0.7255	0.6441	4.6514
0.6439	0.6756	0.1119	3.1121
0.6080	0.6736	0.0059	2.7876
0.6058	0.6736	1.9371e-05	2.7694
0.6058	0.6736	2.0890e-10	2.7694
0.6058	0.6736	-1.1102e-16	2.7694
0.6058	0.6736	-1.1102e-16	2.7694
0.6058	0.6736	-1.1102e-16	2.7694

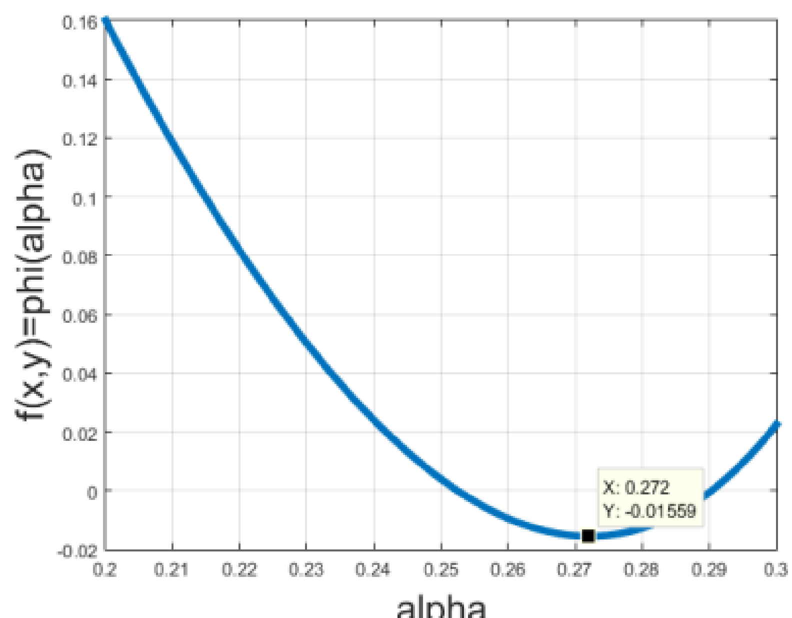
Ex3: Line Search $X_0=(1,1)^T$

$f(x,y) = x^4+xy+y^2$ $f_x = 4x^3+y$ $f_y = x+2y$

Gradient at X_0 = $\binom{5}{3}$

New Position X_1 = $\binom{1}{1} - \alpha\binom{5}{3} = \binom{1-5\alpha}{1-3\alpha}$

$\Phi(\alpha)$ = f(X_1) = $(1-5\alpha)^4+(1-5\alpha)(1-3\alpha)+(1-3\alpha)^2$



Ex3: Line Search $X_0=(1,1)^T$

$f(x,y) = x^4+xy+y^2$

$f_x = 4x^3+y$

$f_y = x+2y$

	fx	fy	α	x	y
1	5	3	0.2721	-0.3606	0.1836
2	-0.0040	0.0066	1.0032	-0.3566	0.1770
3	-0.0045	-0.0027	0.3955	-0.3549	0.1780
4	-7.2371e-04	0.0012	1.0128	-0.3541	0.1768
5	-8.3974e-04	-5.0392e-04	0.3972	-0.3538	0.1770
6	-1.3802e-04	2.3000e-04	1.0146	-0.3537	0.1768
7	-1.6110e-04	-9.6683e-05	0.3976	-0.3536	0.1768
8	-2.6553e-05	4.4238e-05	1.0151	-0.3536	0.1768
9	-3.1020e-05	-1.8622e-05	0.3976	-0.3536	0.1768
10	-5.1118e-06	8.5225e-06	1.0148	-0.3536	0.1768

Ex4: Newton Raphson $X_0=(-2,-2)^T$

$f(x,y) = x^4+xy+y^2$

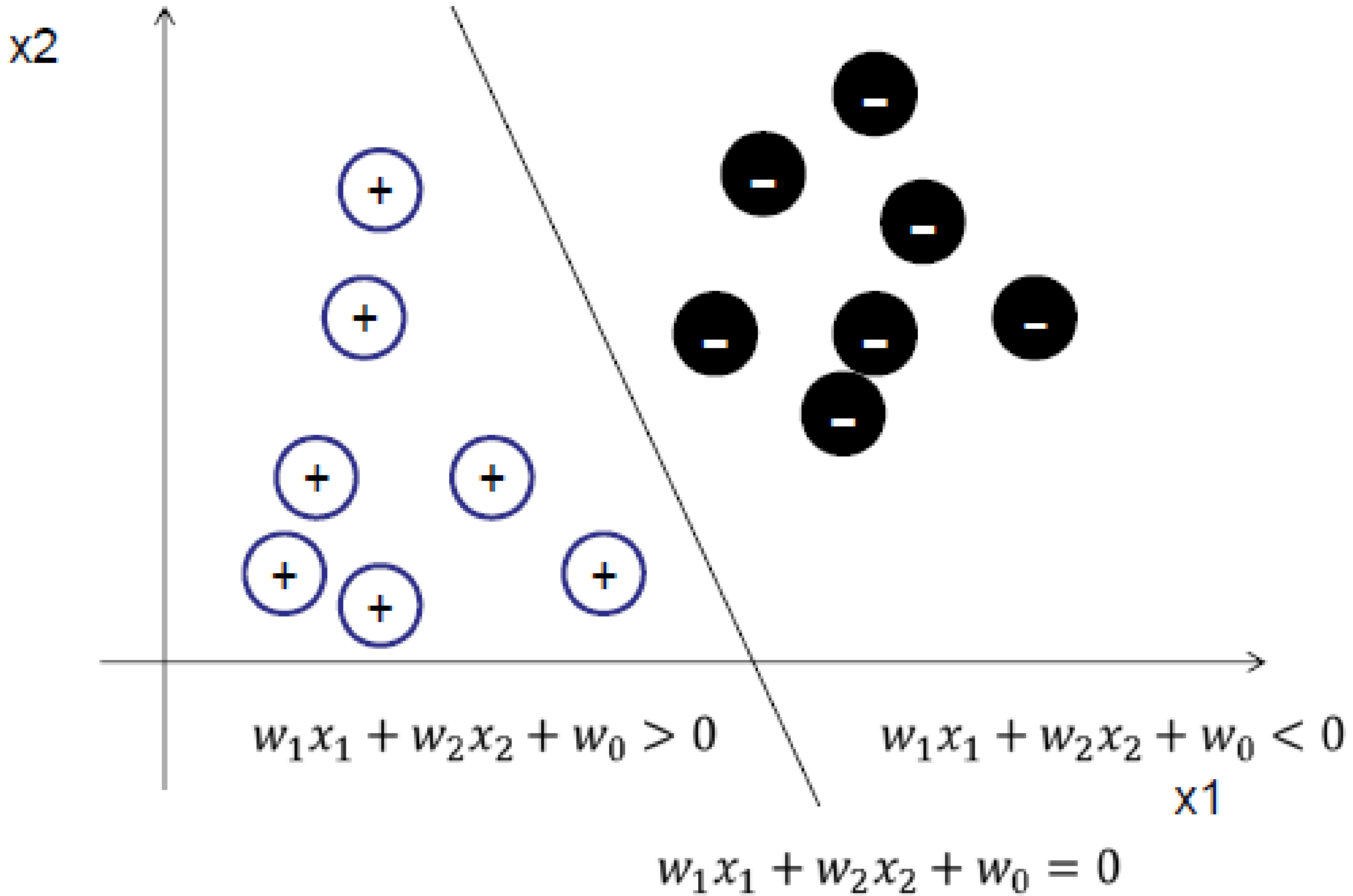
$f_x = 4x^3+y$

$f_y = x+2y$

x	y	f(x,y)	fx	fy	fxx	fxy	fyx	fyy
-1.3474	0.6737	2.8418	-9.1104	6.6613e-16	21.7848	1	1	2
-0.9193	0.4597	0.5031	-2.6484	-1.1102e-16	10.1424	1	1	2
-0.6447	0.3223	0.0688	-0.7494	0	4.9873	1	1	2
-0.4777	0.2388	-0.0050	-0.1971	0	2.7381	1	1	2
-0.3896	0.1948	-0.0149	-0.0417	0	1.8214	1	1	2
-0.3580	0.1790	-0.0156	-0.0045	0	1.5380	1	1	2
-0.3536	0.1768	-0.0156	-8.1783e-05	0	1.5007	1	1	2
-0.3536	0.1768	-0.0156	-2.8342e-08	0	1.5000	1	1	2
-0.3536	0.1768	-0.0156	-3.4139e-15	0	1.5000	1	1	2
-0.3536	0.1768	-0.0156	-2.7756e-17	0	1.5000	1	1	2

Back to Perceptron

Linear Classifier



Canonical Representation

- Each example described by a d-dim vector $\mathbf{x} = [x_1, \ldots, x_d]^T \in R^d$, and assigned a class label $y \in \{-1, 1\}$
- We learn a linear function $g(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d$
- Given unlabeled example $\mathbf{x}$, it predicts
  - $y = 1$ if $g(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d \geq 0$, and $y = -1$ otherwise
- Introduce a dummy feature $x_0 = 1$ for all examples:

$g(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w} = [w_0, w_1, \ldots, w_d]$ and $\mathbf{x} = [x_0, x_1, \cdots, x_d]^T$

- The classifier can be compactly represented by:

$$h(\mathbf{x}) = \operatorname{sign}\left(g(\mathbf{x}, \mathbf{w})\right)$$

- Given its true label y, our prediction for $\mathbf{x}$ is correct if $yg(\mathbf{x}, \mathbf{w}) > 0$
- Goal of learning: find a good **w**
  - such that $h(\mathbf{x})$ makes as few mis-predictions as possible

Learning W: An Optimization Problem

- Formulate learning problem as an optimization problems
  - Given: A set of *N* training examples

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, yn)\}$$

  - Design a loss function *L*
  - Find the weight vector $\mathbf{w}$ that minimizes the expected/average loss on training data

$$J(\mathbf{w}) = \frac{1}{n} \sum_{m=1}^{n} L(\mathbf{w}^T \mathbf{x}_m, y_m)$$

Loss Function

- 0/1 Loss function: $J_{0/1}(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} L(sign(\mathbf{w}^T\mathbf{x}_m), y_m)$

  where $L(y', y) = 0$ when $y' = y$, otherwise $L(y', y) = 1$

- Issue: does not produce useful gradient since the surface of J is piece-wise flat

- Instead we will consider the "**perceptron criterion**"

$$J_p(w) = \frac{1}{n}\sum_{m=1}^{n} \max(0, -y_m\mathbf{w}^T\mathbf{x}_m)$$

- The term $\max(0, -y_m\mathbf{w}^T\mathbf{x}_m)$ is 0 when y_m is predicted correctly, otherwise it is equal to $|\mathbf{w}^T\mathbf{x}_m|$, which can be viewed as the "confidence" in the mis-prediction
- Has a nice gradient leading to the solution region

Stochastic Gradient Descent

- The objective function consists of a sum over data points---we can update the parameter after observing each example
- This is the Stochastic gradient descent approach

$$J(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

$$J_m(\mathbf{w}) = \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

$$\nabla J_m = \begin{cases} 0 & \text{if } \; y_m \mathbf{w} \cdot \mathbf{x}_m > 0 \\ -y_m \mathbf{x}_m & \text{otherwise} \end{cases}$$

After observing $(\mathbf{x}_m, y_m)$, if it is a mistake $\mathrm{w} \leftarrow \mathrm{w} + y_m \mathbf{x}_m$

Batch Perceptron Algorithm

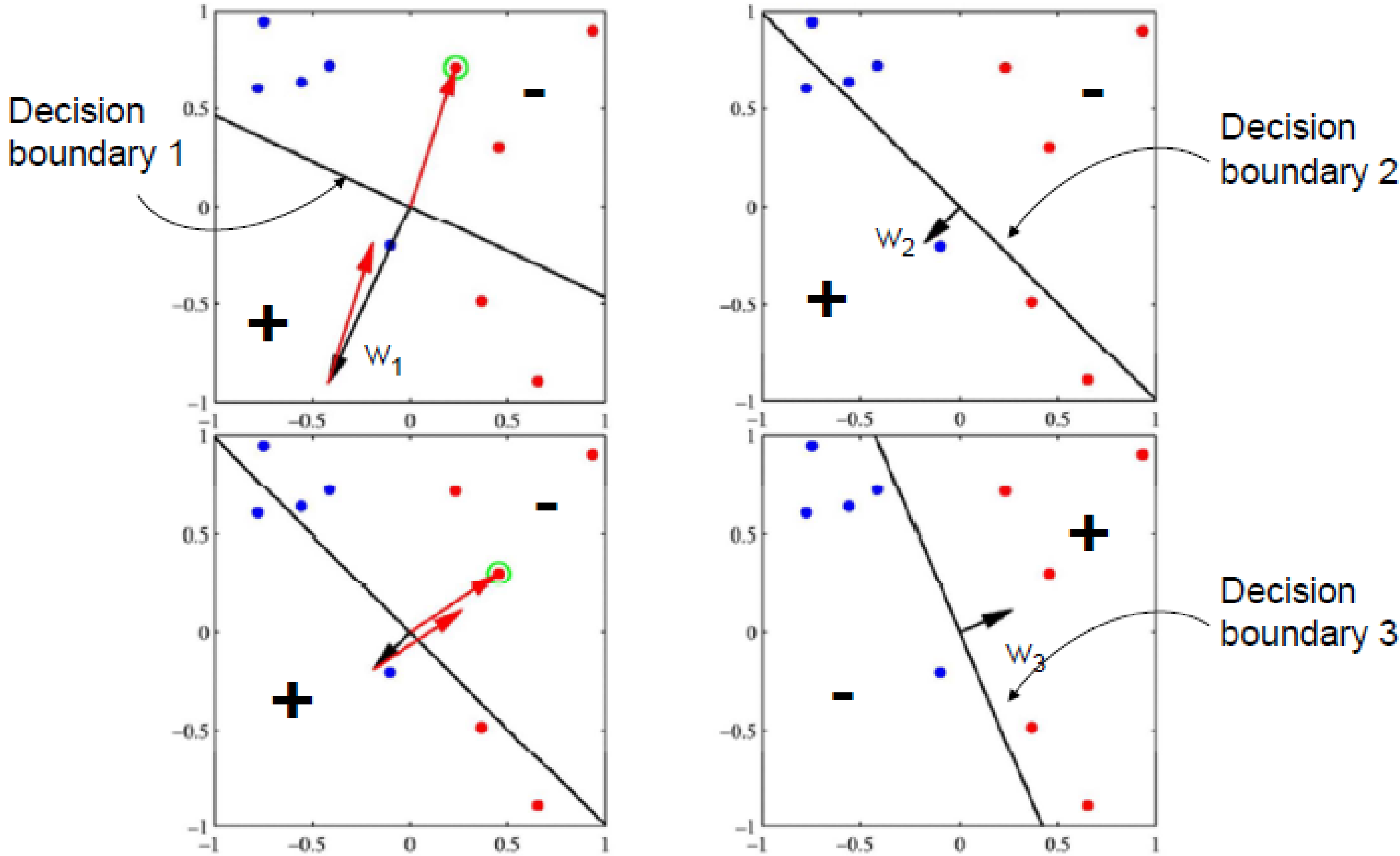
$$
\begin{array}{l}
\text{Given : training examples } (\mathbf{x}_m\ ,\ y_m),\ m = 1,\ldots,n \\
\text{Let } \mathbf{w} \leftarrow (0,0,0,\ldots,0) \\
\text{do} \\
\qquad delta \leftarrow (0,0,0,\ldots,0) \\
\qquad \text{for } m\ =\ 1 \text{ to } n \text{ do} \\
\qquad\qquad u_m \leftarrow \mathbf{w} \cdot \mathbf{x}_m \\
\qquad\qquad \text{if } y_m \cdot u_m\ \le\ 0 \\
\qquad\qquad\qquad delta \leftarrow delta - y_m \cdot x_m \\
\qquad delta \leftarrow delta \,/\, n \\
\qquad \mathbf{w} \leftarrow \mathbf{w} - \lambda\, delta \\
\text{until } |\, delta \,| < \varepsilon
\end{array}
$$

Simplest case: λ = *1* and don't normalize – 'Fixed increment perceptron'

λ – the step size

- Also referred as the learning rate
- In practice, recommend to decrease η as learning continues
- Some optimization approaches set step-size automatically, e.g., by line search, and converge faster
- If linearly separable, there is only one basin for the hinge loss, thus local minimum is the global minimum

When an error is made, moves the weight in a direction that corrects the error



Red points belong to the positive class,
blue points belong to the negative class

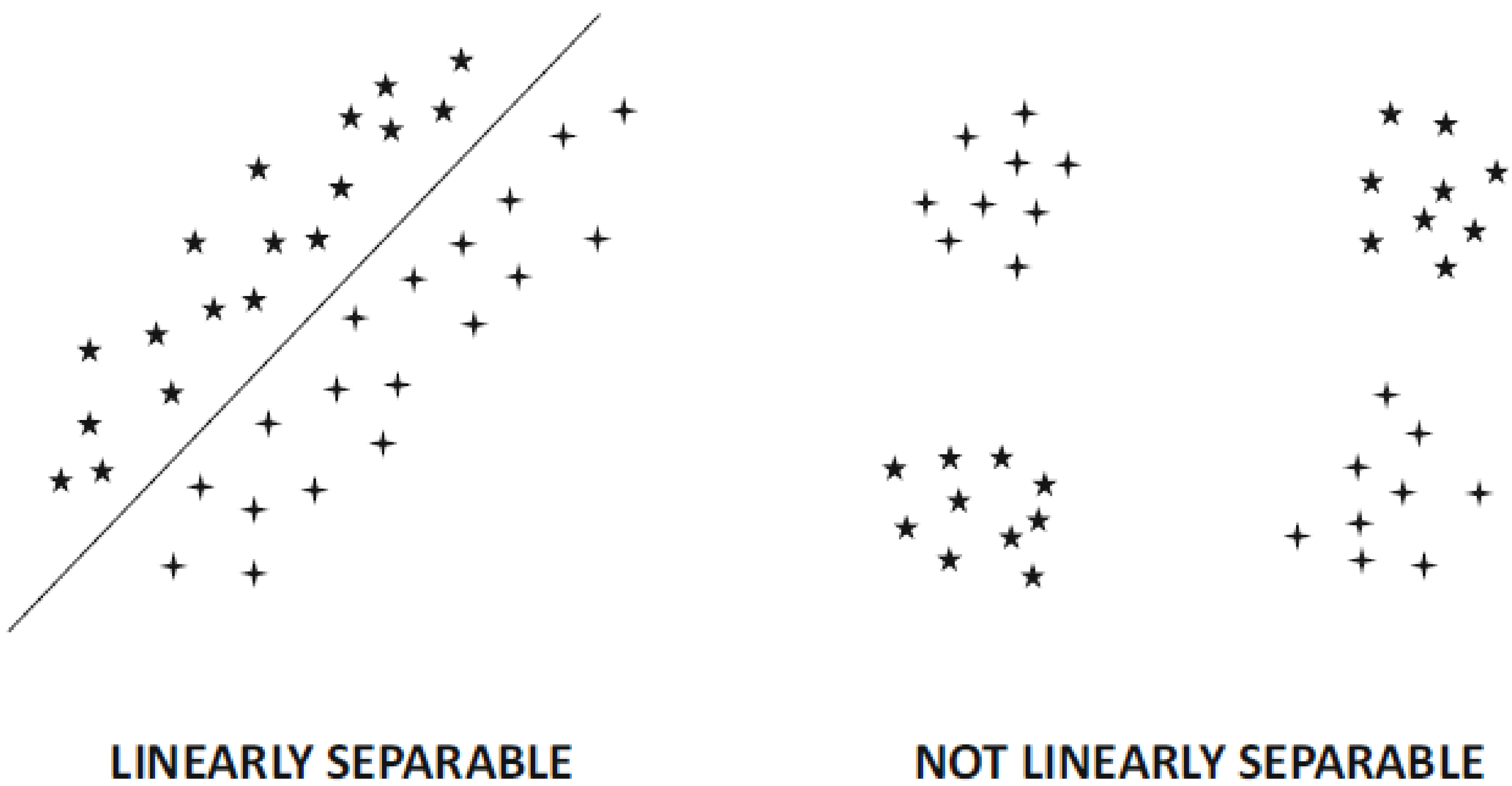
Linear Classifier

- Simplest model – fewer parameters to learn (requires less training data to learn reliably)

- Intuitively appealing -- draw a straight line (for 2-d inputs) or a linear hyper-plane (for higher dimensional inputs) to separate positive from negative

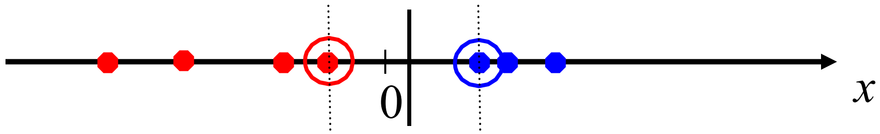
- Can be used to learn nonlinear models as well. How?
  - Introducing nonlinear features (e.g., $x_1^2, x_2^2, x_1 x_2 \ldots$)
  - Use kernel tricks (we will talk about this later this term)

Will it work?

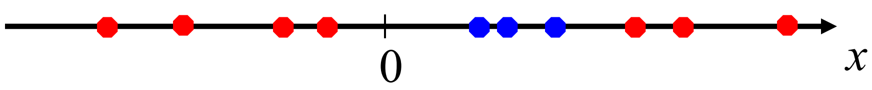


1D Example

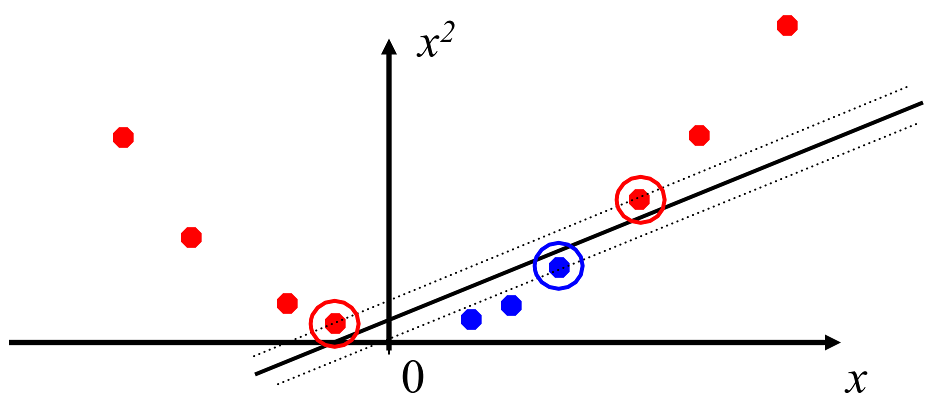
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?



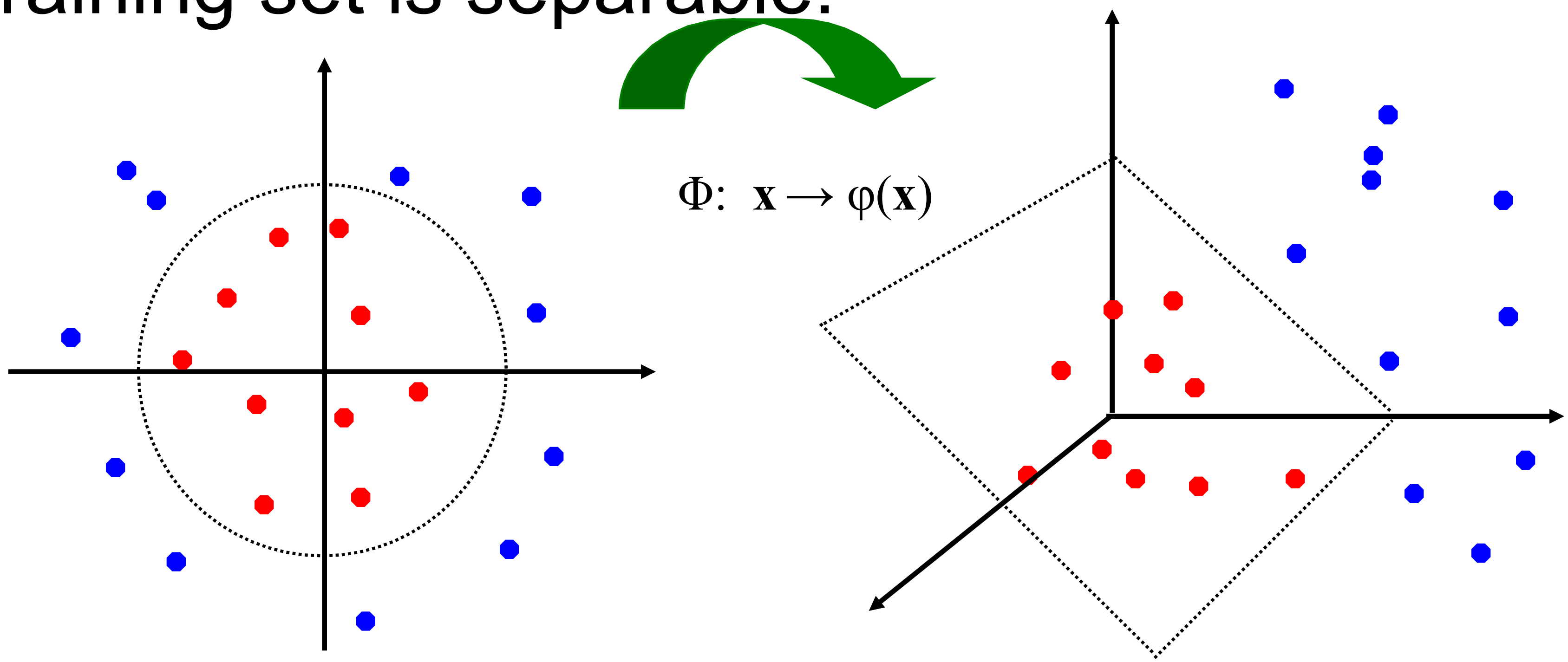
- We can map it to a higher-dimensional space:



Andrew Moore

2D Example

Map the original input space to some higher -dimensional feature space where the training set is separable:

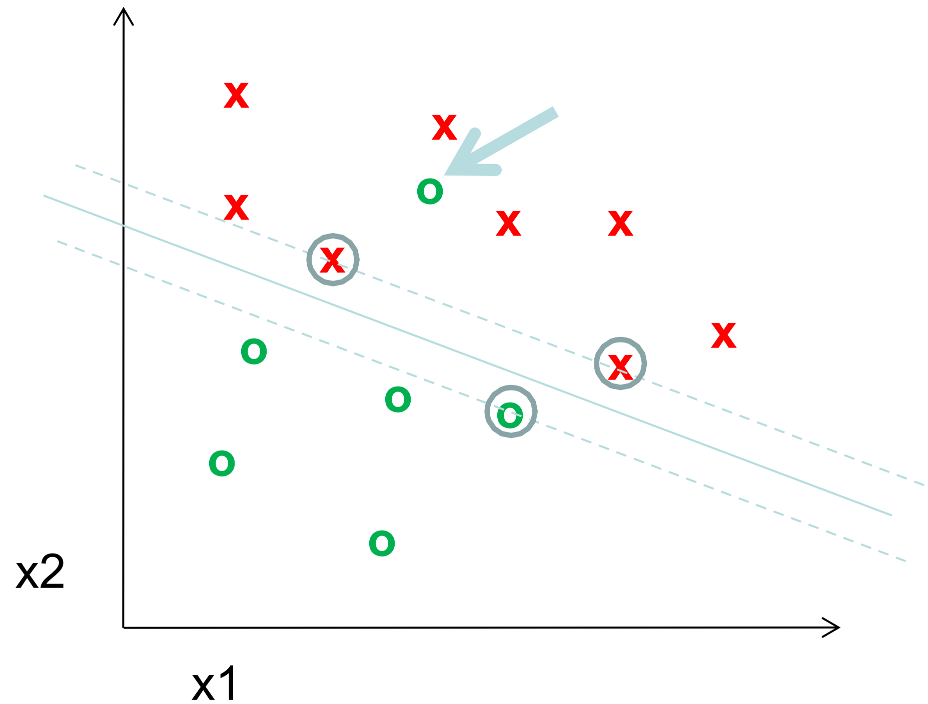


Slide credit: Andrew Moore

Relation to SVM

Find a *linear function* to separate the classes:

$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$



What if my data are not linearly separable?

Introduce flexible ‘hinge’ loss (or ‘soft-margin’)

Relation to SVM: Modified Loss Function

$$J(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} \max(0,\ 1 - y_m \mathbf{w}^T \mathbf{x}_m)$$

Introduce flexible 'hinge' loss (or 'soft-margin')

Relation to SVM: Modified Loss Function

Does this affect the training algorithm?

Demo