Pushing Regression Further: Polynomial Regression

Alexander G. Ororbia II

Introduction to Machine Learning

CSCI-335

3/16/2026

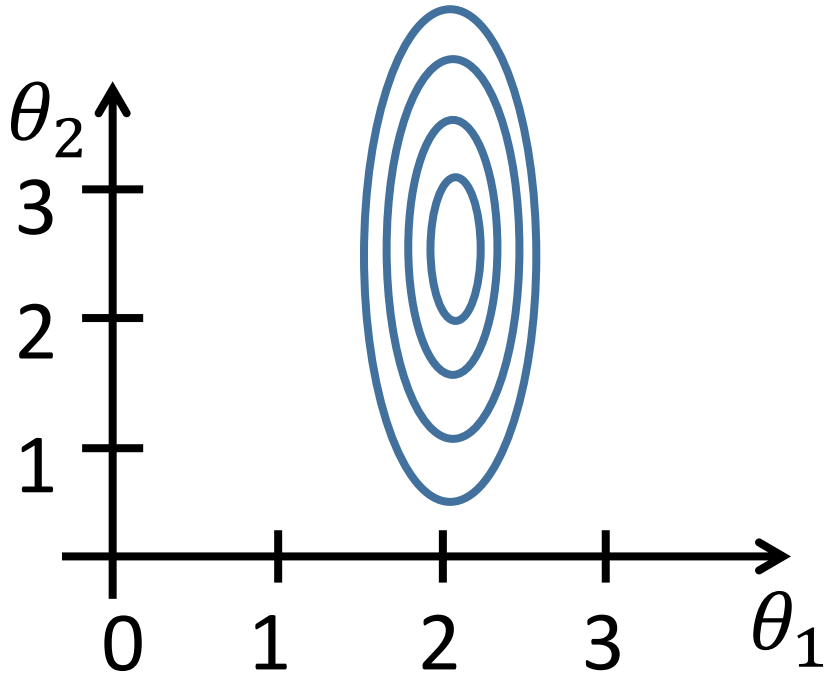
Special Thanks: Sargur N. Srihari, Andrew Ng, Carlos Guestrin

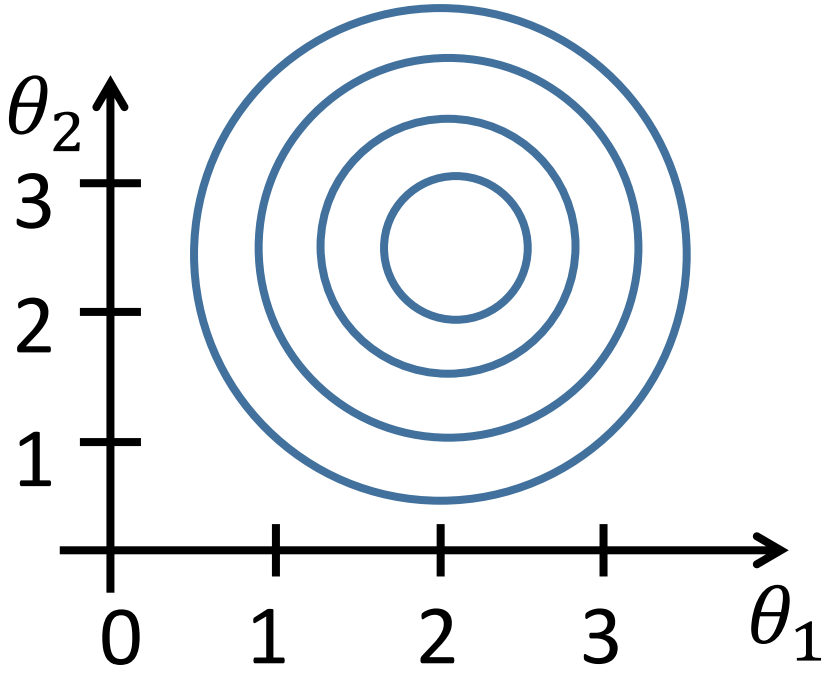
Gradient Descent in Practice: Feature Scaling

- *Idea*: Make sure features are on a similar scale (e.g,. $-1 \leq x_i \leq 1$)
- *Example*: $x_1 =$ size (0-2000 feat^2)

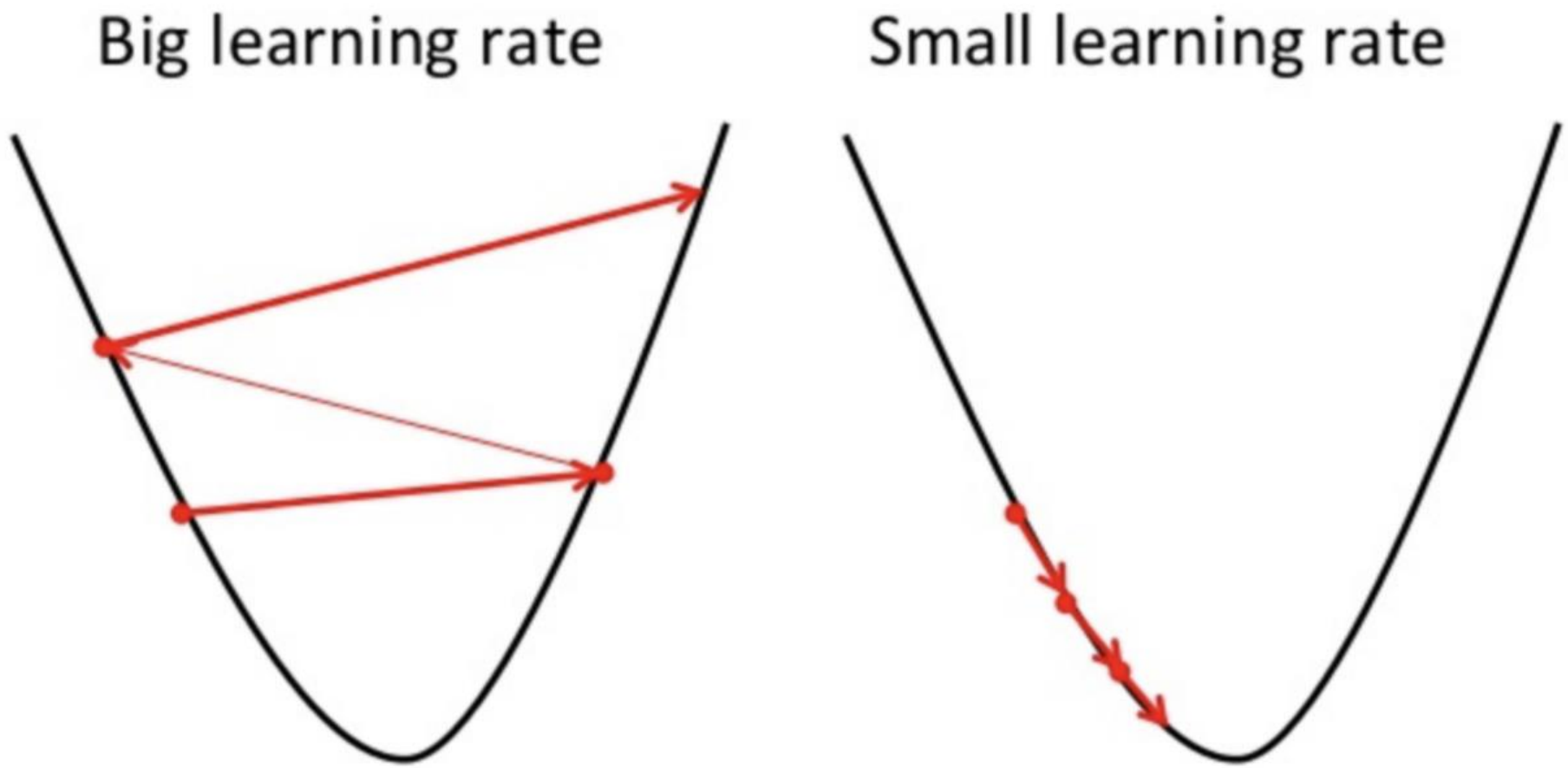
  $x_2 =$ number of bedrooms (1-5)

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i}$$





Gradient Descent Learning Rate / Step-Size

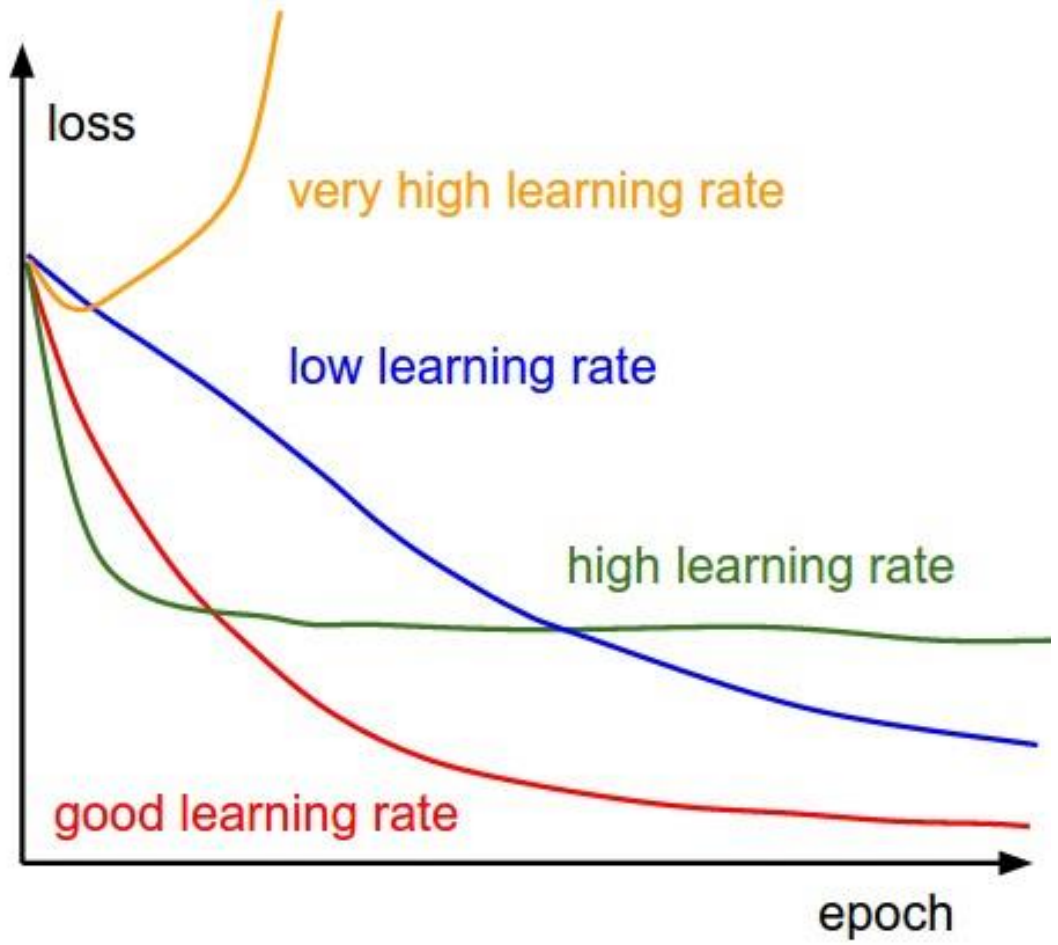


Gradient Descent in Practice: Step Size

- Automatic convergence test
- α too small: slow convergence
- α too large: may not converge

- To choose α, try

0.001, … 0.01, …, 0.1, … , 1



Learning rate is also the “step size” in statistics/mathematics

Image credit: CS231n@Stanford

A Simple Approach to Curve Fitting

Representation

- Fit the data using a *polynomial function*

$$y(x, \mathrm{w}) = w_0 + w_1 x + w_2 x^2 + .. + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

  - where M is the order of the polynomial
- Is higher value of M better? We'll see shortly!
- Coefficients $w_0, \ldots w_M$ are collectively denoted by vector $\mathbf{w}$
- It is a nonlinear function of x, but a linear function of the unknown parameters

(Note: still a linear model; linear in parameter space)

Representation + Evaluation

Polynomial Curve Fitting

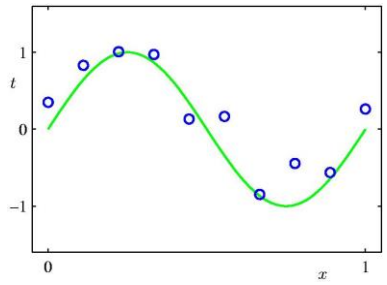
– With a single input variable x

– $y(x,\mathbf{w}) = w_0+w_1x+w_2x^2+\ldots+w_Mx^M = \sum_{j=0}^{M} w_j x^j$

M = order of polynomial,

x^j denotes x raised to power j,

Coefficients $w_0,\ldots,w_M$ are collectively denoted by vector $\mathbf{w}$



Training data set
N=10, Input x, target t

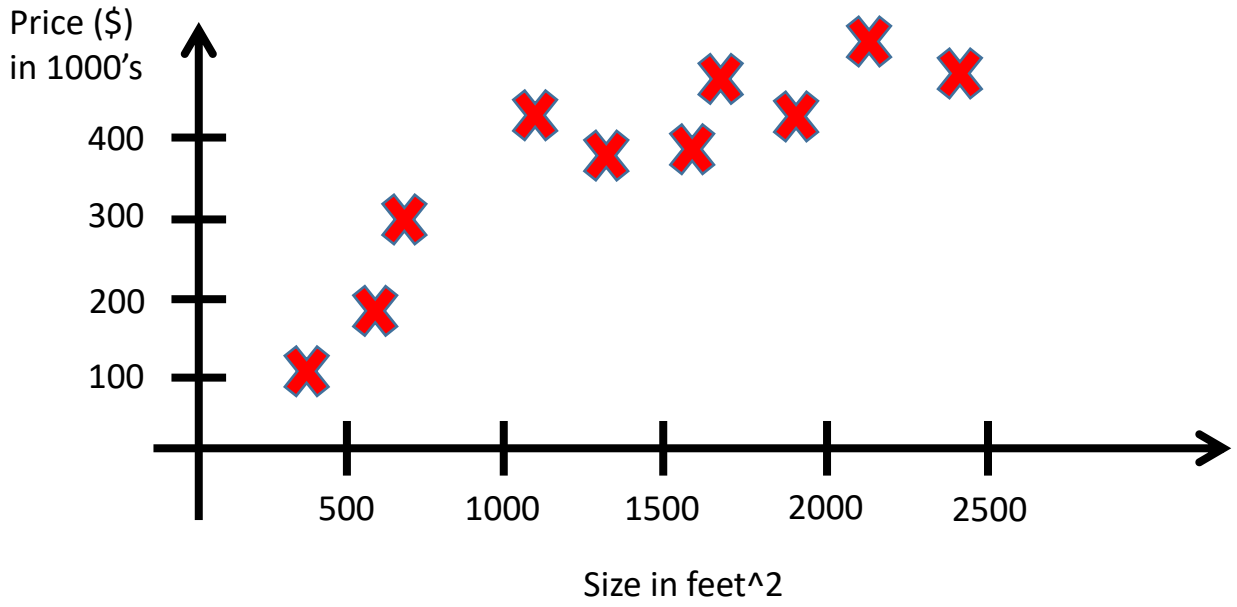
– **Task**: Learn $\mathbf{w}$ from training data $D=\{(x_i,t_i)\}$, $i=1,..,N$

- Can be done by minimizing an error function that minimizes misfit between $y(x,\mathbf{w})$ for any given $\mathbf{w}$ and training data
- One simple choice of error function is sum of squares of error (*SSE*) between predictions $y(x_n,\mathbf{w})$ for each data point x_n and corresponding target values t_n so that we minimize: $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w})-t_n\}^2$
- It is zero when function $y(x,\mathbf{w})$ passes exactly through each training data point

Optimization = gradients of evaluation (for SGD)

A Polynomial Hypothesis

Apply same principles of linear regression derivation for each parameter (just change index j > 1)



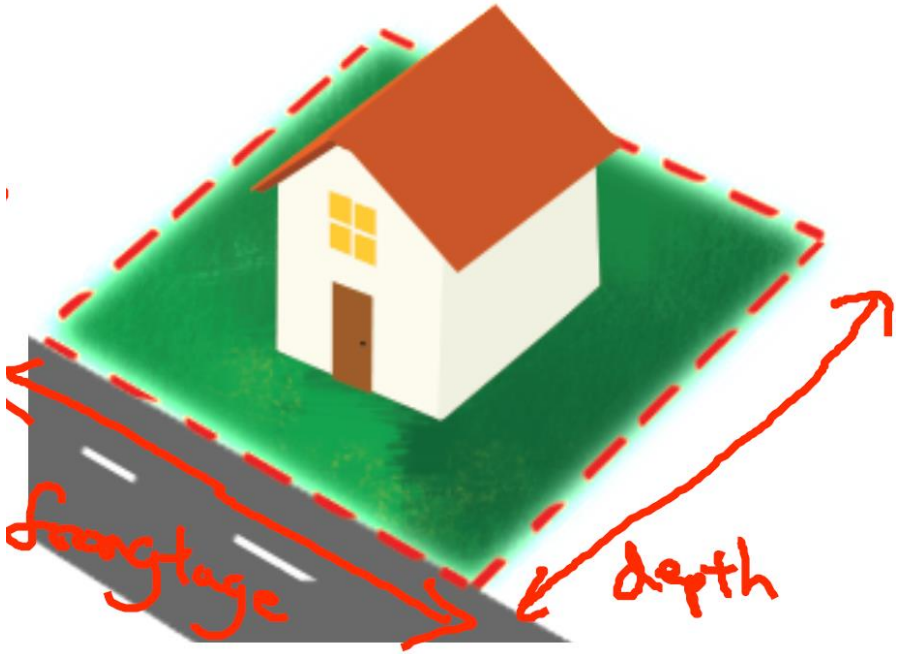
x_1 = (size)
x_2 = (size)^2
x_3 = (size)^3

- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

$$= \theta_0 + \theta_1(size) + \theta_2(size)^2 + \theta_3(size)^3$$

House Price(s) Prediction

- $h_\theta(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$
- Area: $xy =$ frontage (x) $\times$ depth (y)
- Housing model: $h_\theta(x) = \theta_0 + \theta_1 xy$
  (a multiplicative feature interaction)



On Basis Functions

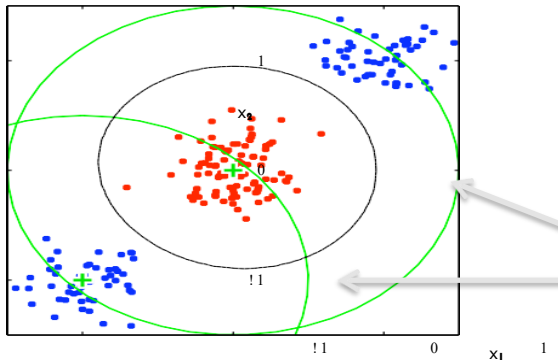
- In many applications, we apply some form of fixed-preprocessing, or feature extraction, to the original data variables
- If the original variables comprise the vector $\mathbf{x}$, then the features can be expressed in terms of basis functions $\{\phi_j(\mathbf{x})\}$
  - By using nonlinear basis functions we allow the function $y(\mathbf{x},\mathbf{w})$ to be a nonlinear function of the input vector $\mathbf{x}$
    - They are linear functions of parameters (gives them simple analytical properties), yet are nonlinear wrt input variables

Fixed Basis Functions

Although we use linear (classification) models

Linear-separability in ***feature*** space *does not* imply linear-separability in ***input*** space

Original Input Space (x_1, x_2)

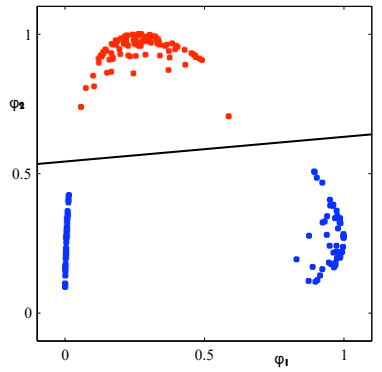


not linearly separable

Nonlinear transformation of inputs using vector of basis functions $\boldsymbol{\phi}(\boldsymbol{x})$

Basis functions are Gaussian with centers shown by crosses and green contours

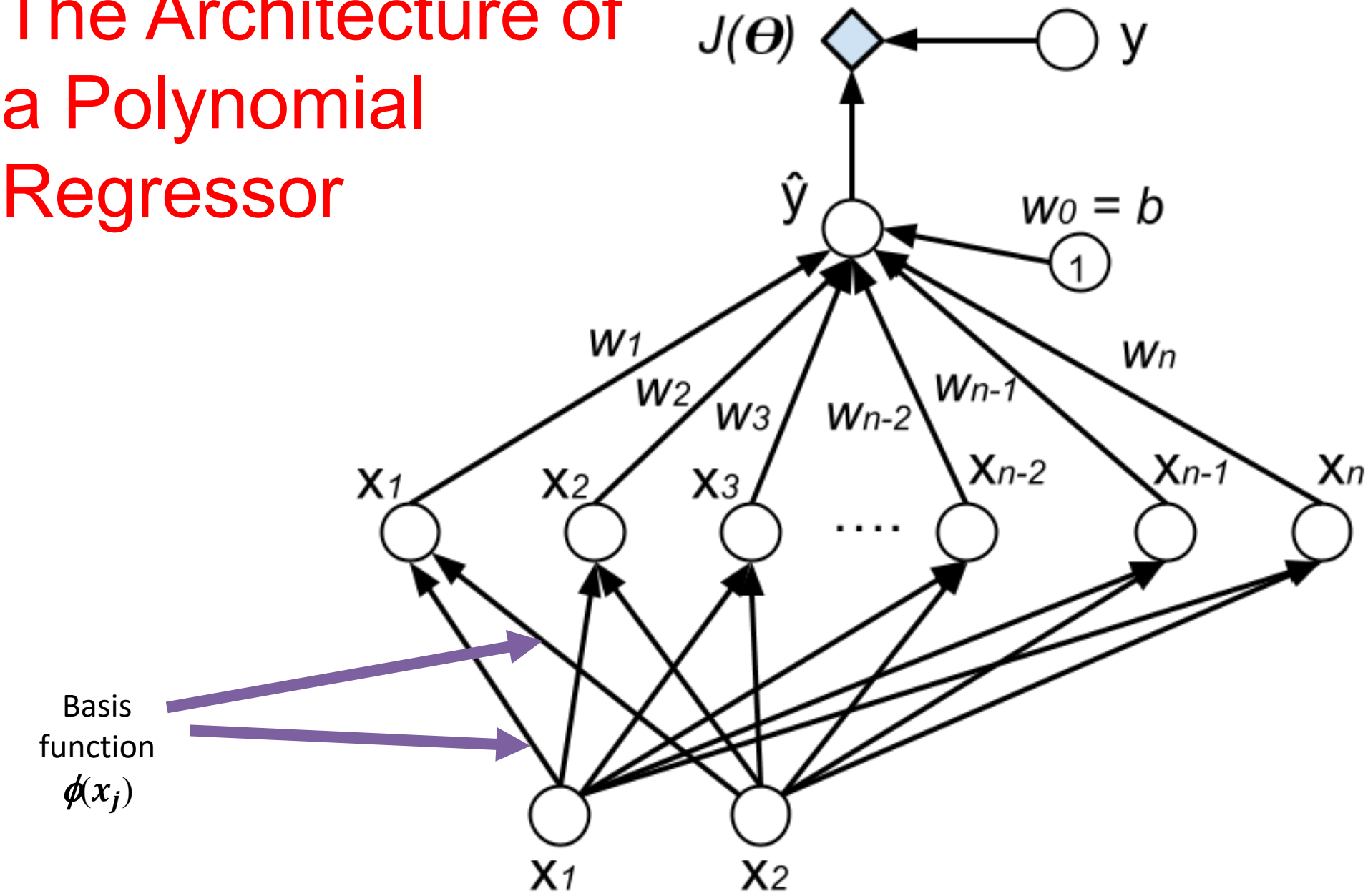
Feature Space (ϕ_1, ϕ_2)



linearly separable

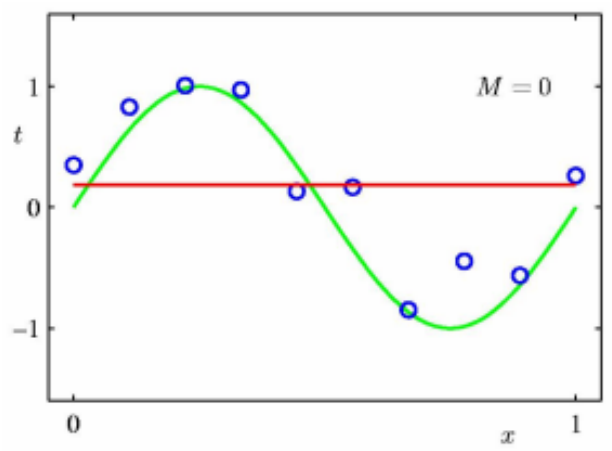
Basis functions with increased dimensionality often used

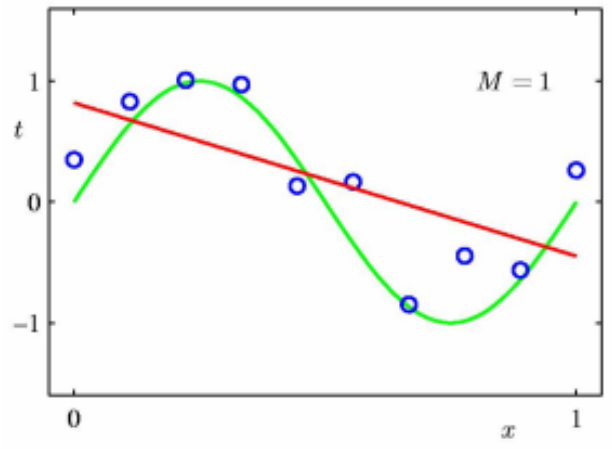
The Architecture of a Polynomial Regressor



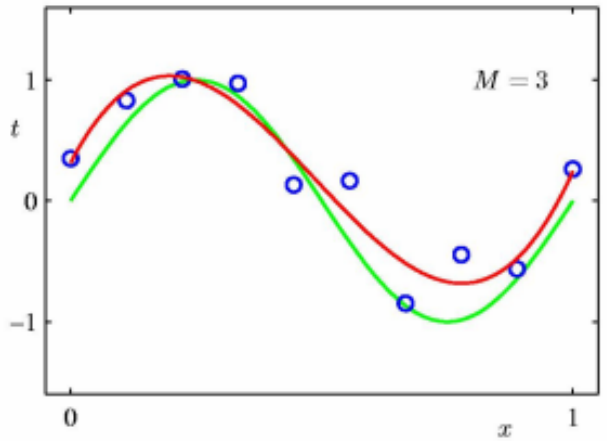
Choosing the Order of *M*

- Model Comparison or Model Selection
- Red lines are best fits with
  - $M = 0,1,3,9$ and $N=10$

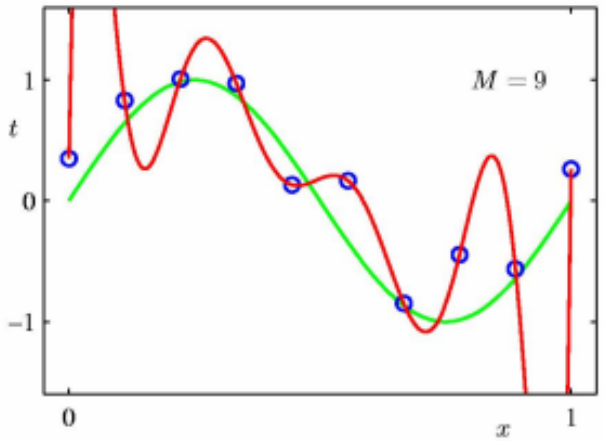




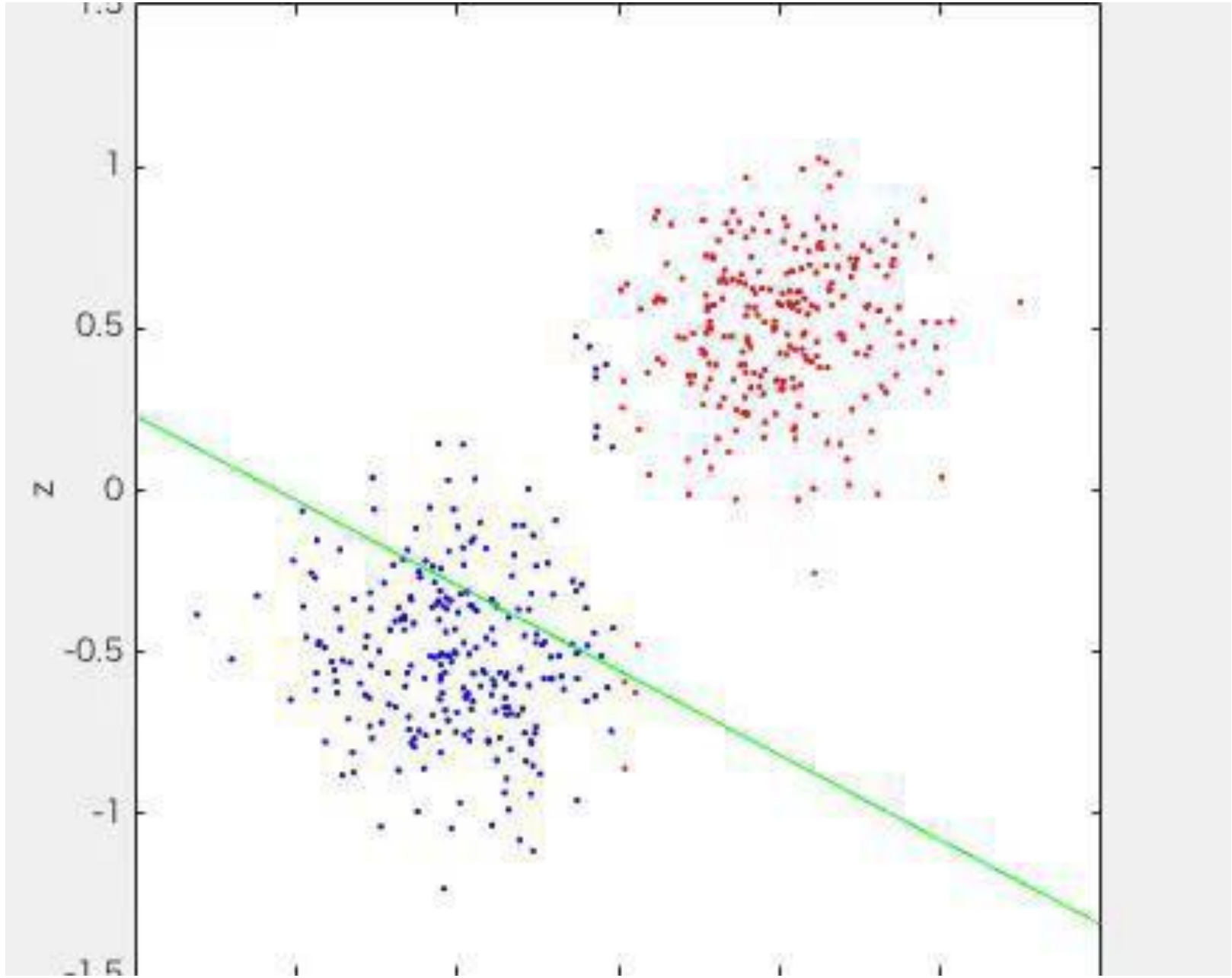
← Poor representations of $\sin(2\pi x)$

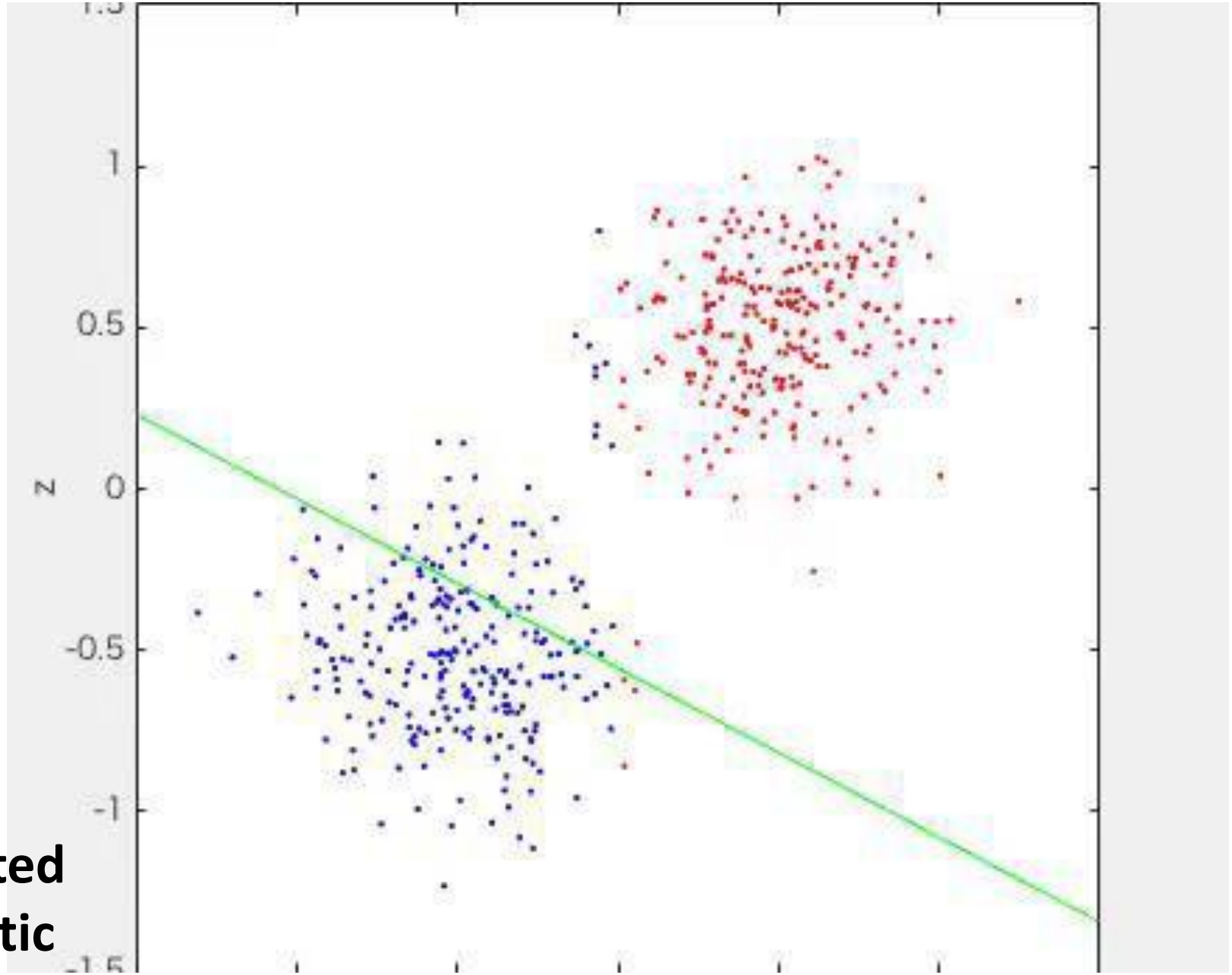


← Best Fit to $\sin(2\pi x)$



Over Fit Poor representation of $\sin(2\pi x)$


1
0.5
z 0
-0.5
-1



But if we wanted some automatic capacity control?

Limiting Model Capacity

- Regularization has been used for decades prior to advent of deep learning
- Linear- and logistic-regression allow simple, straightforward and effective regularization strategies
  - Adding a parameter norm penalty $\Omega(\theta)$ to the objective function J :

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

    - where $\alpha\varepsilon[0,\theta)$ is a hyperparameter that weight the relative contribution of the norm penalty term Ω
      - Setting α to 0 results in no regularization. Larger values correspond to more regularization

Gradient of Regularized Objective

- Objective function (with no bias parameter)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

- Corresponding parameter gradient

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- To perform single gradient step, perform update:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \varepsilon\left(\alpha \boldsymbol{w} + \nabla_w J\left(\boldsymbol{w}; X, \boldsymbol{y}\right)\right)$$

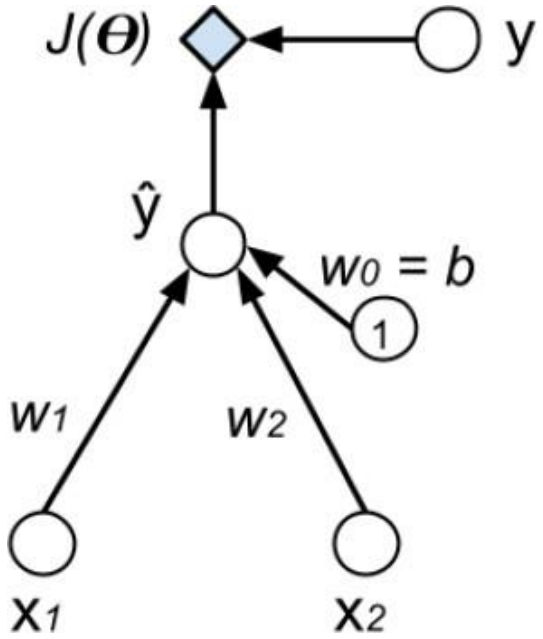
- Written another way, the update is

$$\boldsymbol{w} \leftarrow (1 - \varepsilon\alpha)\boldsymbol{w} - \varepsilon \nabla_w J\left(\boldsymbol{w}; X, \boldsymbol{y}\right)$$

  - We have modified learning rule to shrink $\boldsymbol{w}$ by constant factor $1-\varepsilon\alpha$ at each step

Multivariate Regressor Architecture

$$f_\Theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$ ← **Your Hypothesis!**



Cost:

$$\mathcal{J}(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(f_\Theta(x^i) - y^i)^2 + \frac{\beta}{2m}\sum_{j=1}^{n}\theta_j^2$$

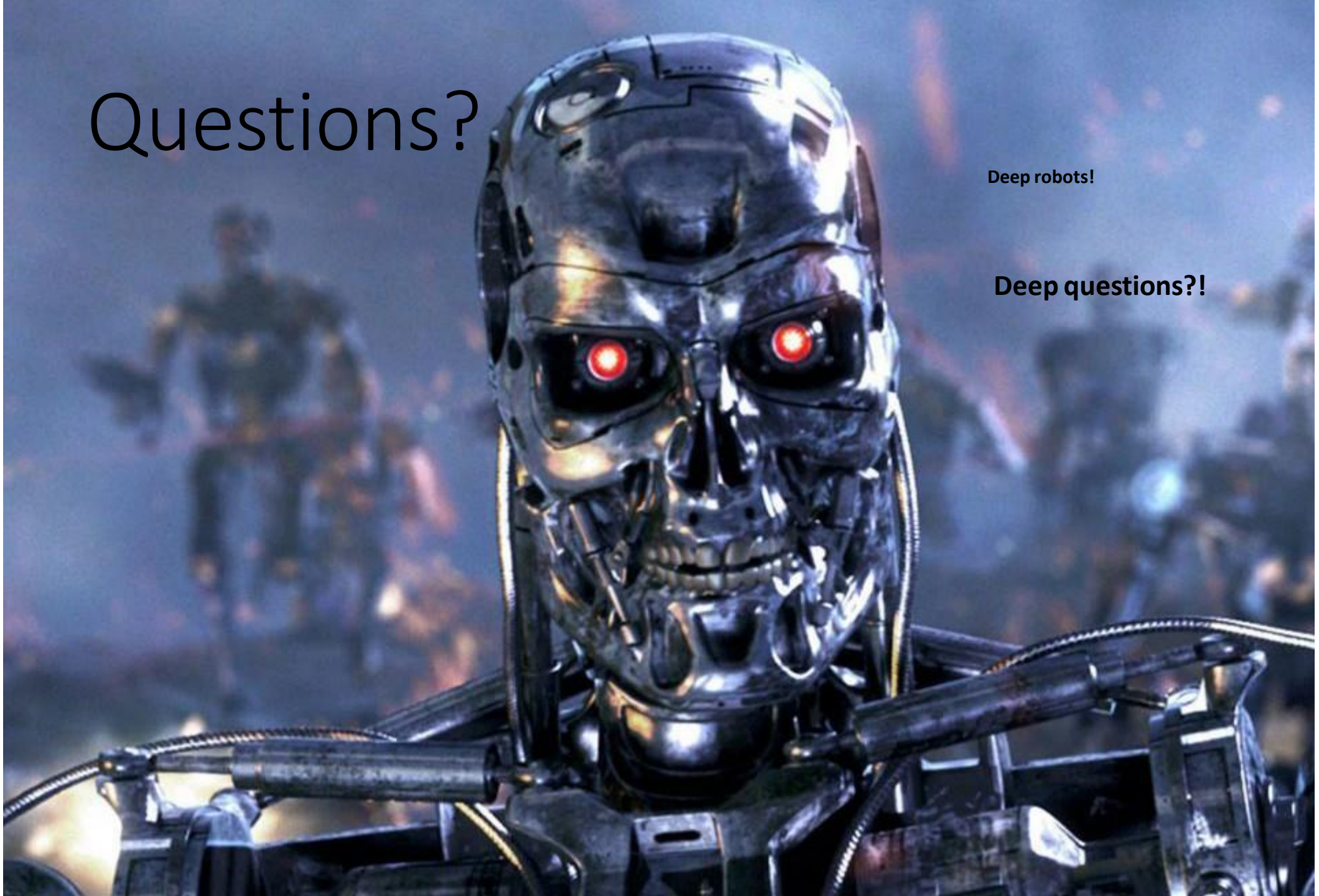
Derivative/Update:

$$\frac{\partial \mathcal{J}(\Theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(f_\Theta(x^i) - y^i)x_j^i + \frac{\beta}{m}\theta_j$$

Optimizer:

$$\theta_j = \theta_j - \alpha\frac{\partial \mathcal{J}(\Theta)}{\partial \theta_j} = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(f_\Theta(x^i) - y^i)x_j^i, j = 0, 1, 2, \cdots, n.$$

Questions?



Deep robots!

Deep questions?!