



On Multivariate Regression (and Complex Interactions)

Alexander G. Ororbia II
Introduction to Machine Learning
CSCI-335
10/5/2026

Gradient Descent for Linear Regression

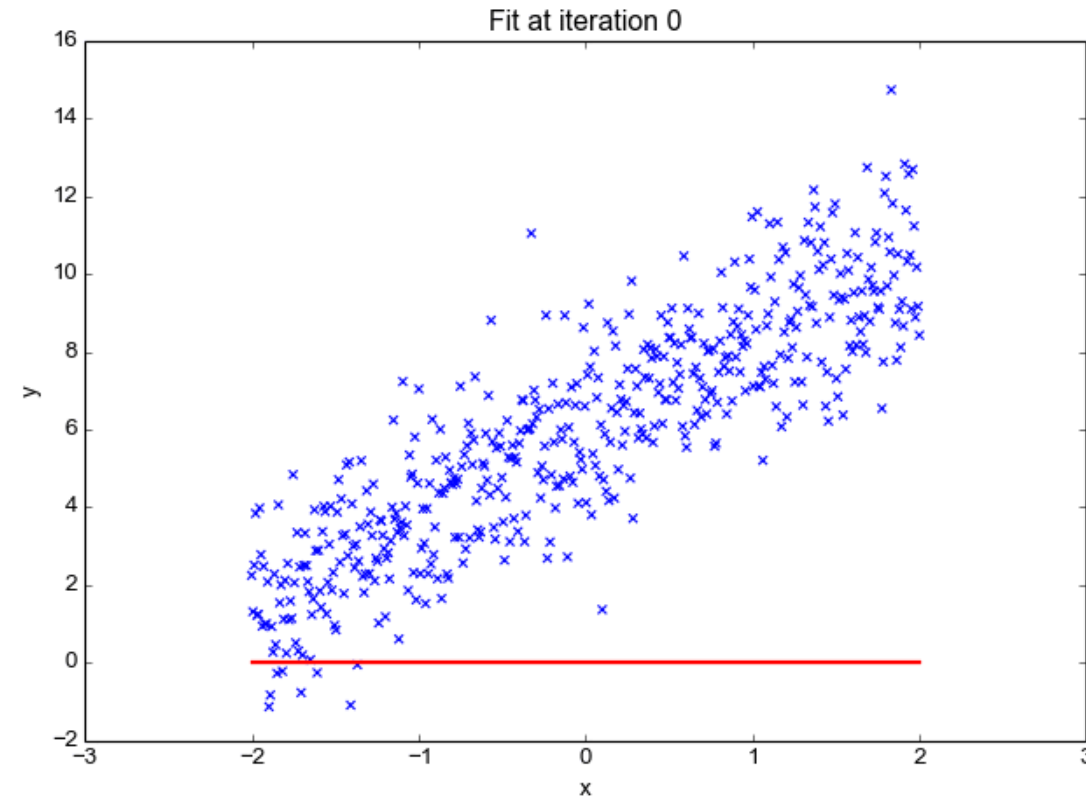
Repeat until convergence{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

}

Update θ_0 and θ_1 *simultaneously*



Batching Up Gradients

- *Another trade-off*: compute/simulation time and gradient error
 - N patterns in a dataset, M used patterns at any time (in a “mini-batch” subset)
 - Batch gradient descent (GD)
 - $m = M = N$
 - Mini-Batch GD
 - $m = M; M < N$, generally $M \ll N$
 - Stochastic GD (SGD)
 - $m = M = 1$

*Involve sampling
randomness!*



Computational Bottleneck

- A recurring problem in machine learning:
 - Large training sets are necessary for good generalization
 - *BUT* large training sets are also computationally expensive to use
- Stochastic gradient descent (SGD) is an extension of gradient descent (GD) that offers a solution
 - Moreover, it is a vehicle for generalization beyond training set
 - Expectation may be approximated using small set of samples (we will also later refer to these sets as “mini-batches” → mini-batch GD)

Linear Regression

- Model representation
- Cost function
- Gradient descent
- **Multivariate regression**

Training Dataset

What if instead
of this univariate
dataset...

Size in feet ² (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Multiple Features (Input Variables)

...we had this
multivariate
dataset instead?

Size in feet ² (x_1)	Number of bedrooms (x_2)	Number of floors (x_3)	Age of home (years) (x_4)	Price (\$) in 1000's (y)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...				...

Notation:

$D = n$ = Number of features

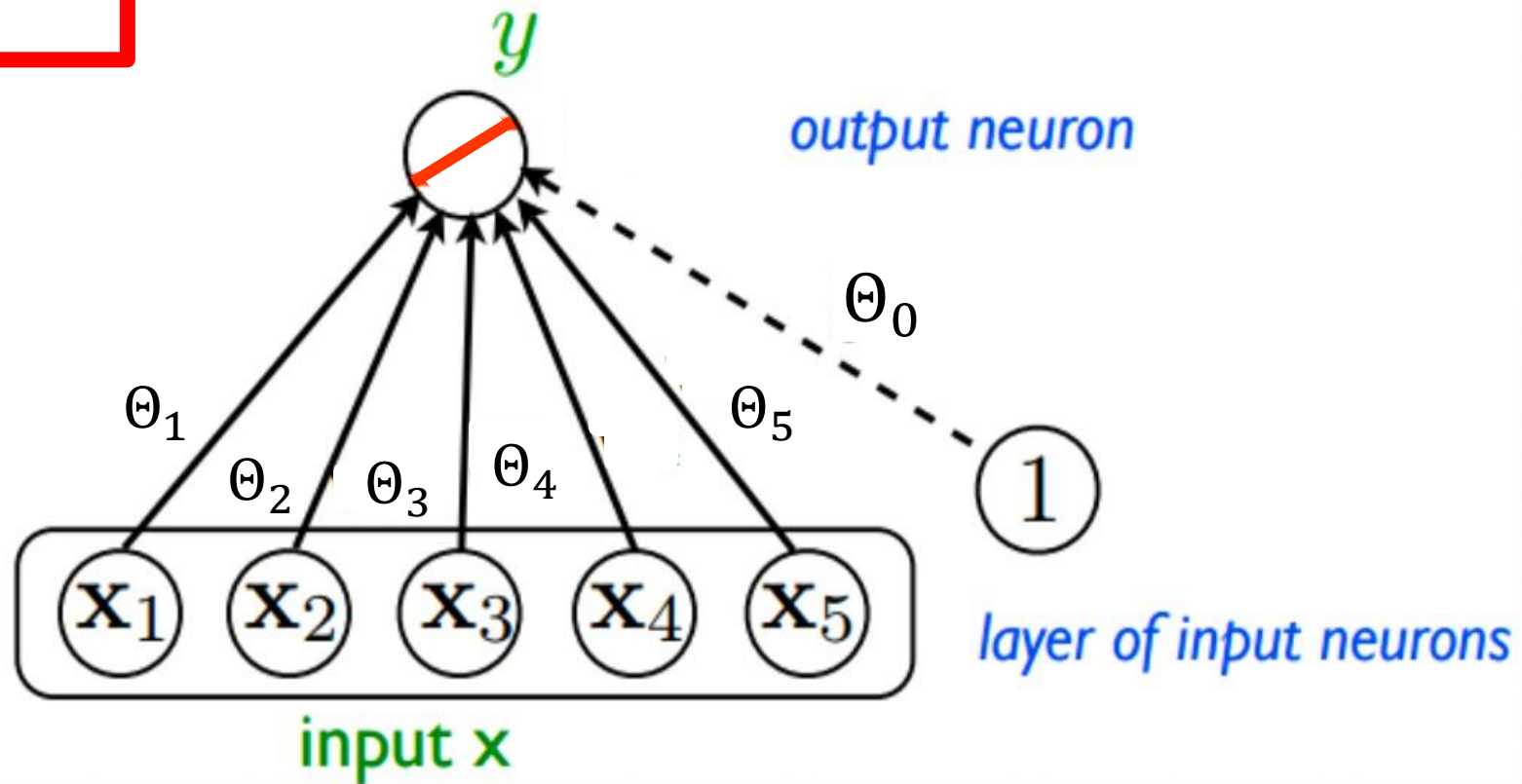
$x^{(i)}$ = Input features of i^{th} training example

$x_j^{(i)}$ = Value of feature j in i^{th} training example

$x_3^{(2)} = ?$

$x_3^{(4)} = ?$

Looks like we're
going to need to
alter our regressor
just a bit...



Hypothesis: Multivariate Form

Previously:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Now:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- For convenience of notation, define $x_0 = 1$
 $(x_0^{(i)} = 1 \text{ for all examples})$

- $\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in R^{n+1}$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in R^{n+1}$$

This yields the "bias"

This is that "padding feature" we talked about last time!

- $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
 $= \boldsymbol{\theta}^T \mathbf{x}$

Notice that this becomes a simple dot-product!

Gradient Descent (Multivariate)

- Previously ($n = 1$)

Repeat until convergence{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

}

- New algorithm ($n \geq 1$)

Repeat until convergence{

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

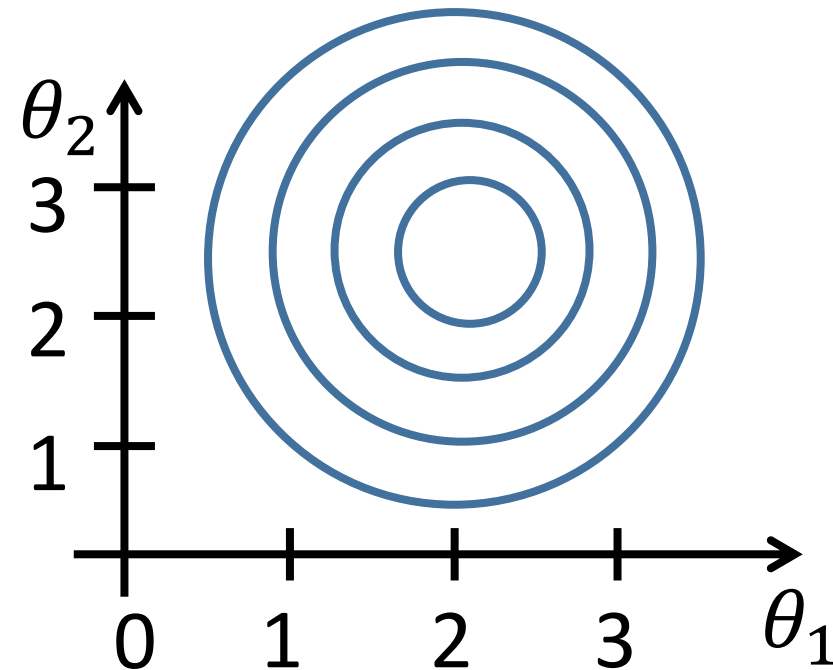
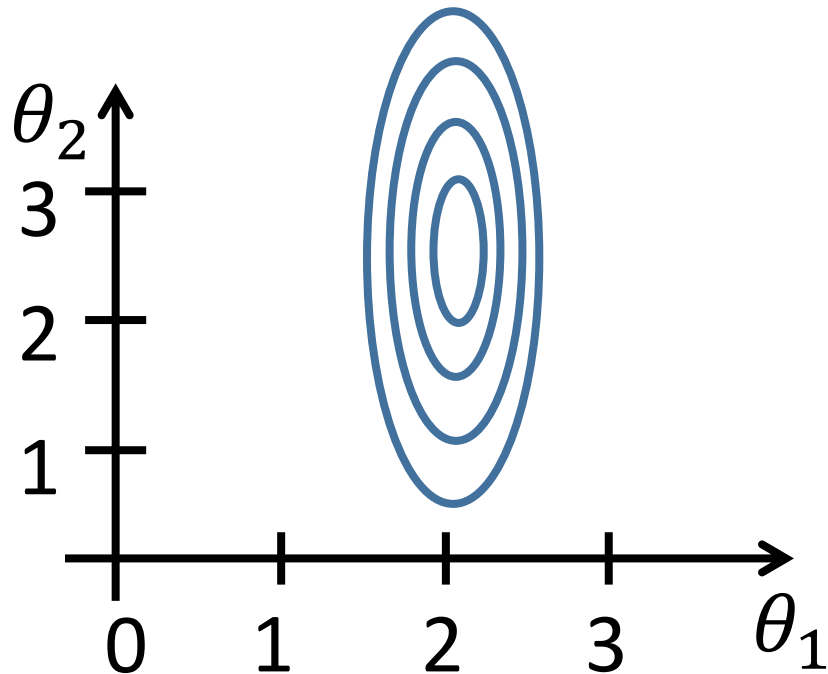
Simultaneously update

θ_j , for $j = 0, 1, \dots, n$

Gradient Descent in Practice: Feature Scaling

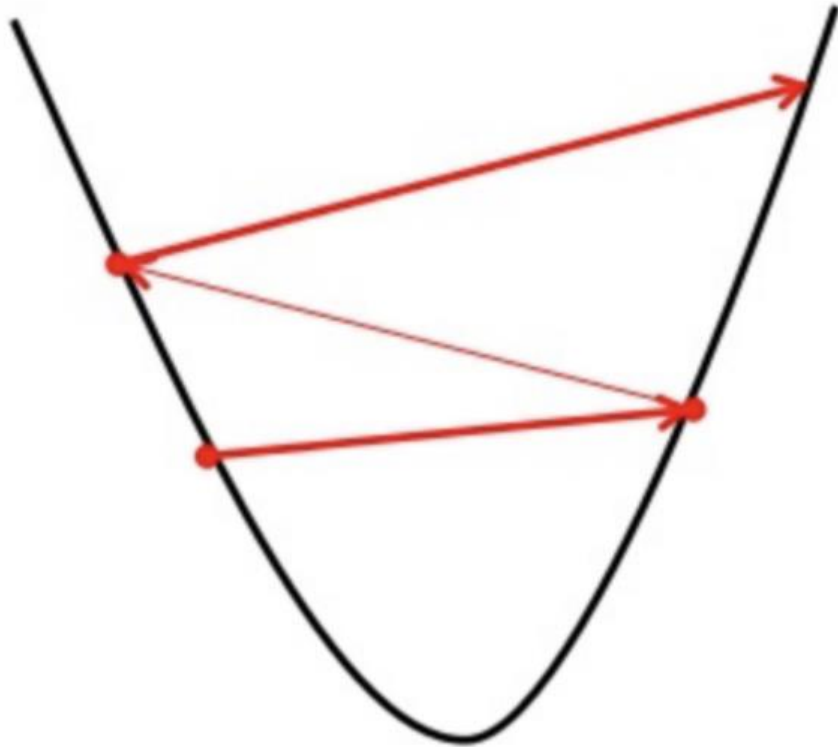
- *Idea*: Make sure features are on a similar scale (e.g., $-1 \leq x_i \leq 1$)
- *Example*: x_1 = size (0-2000 feat²)
 x_2 = number of bedrooms (1-5)

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

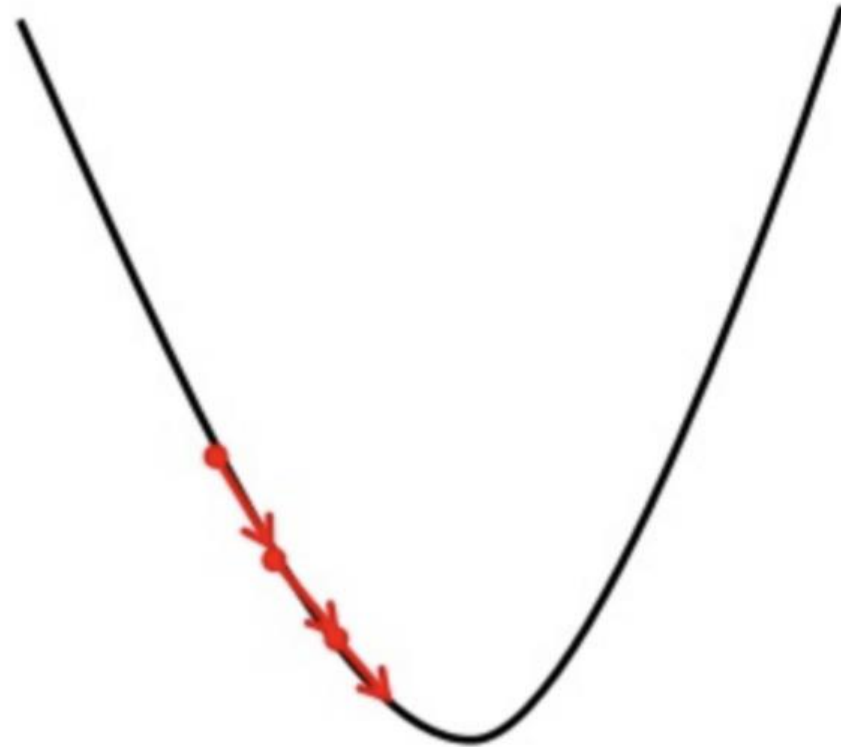


Gradient Descent Learning Rate / Step-Size

Big learning rate



Small learning rate

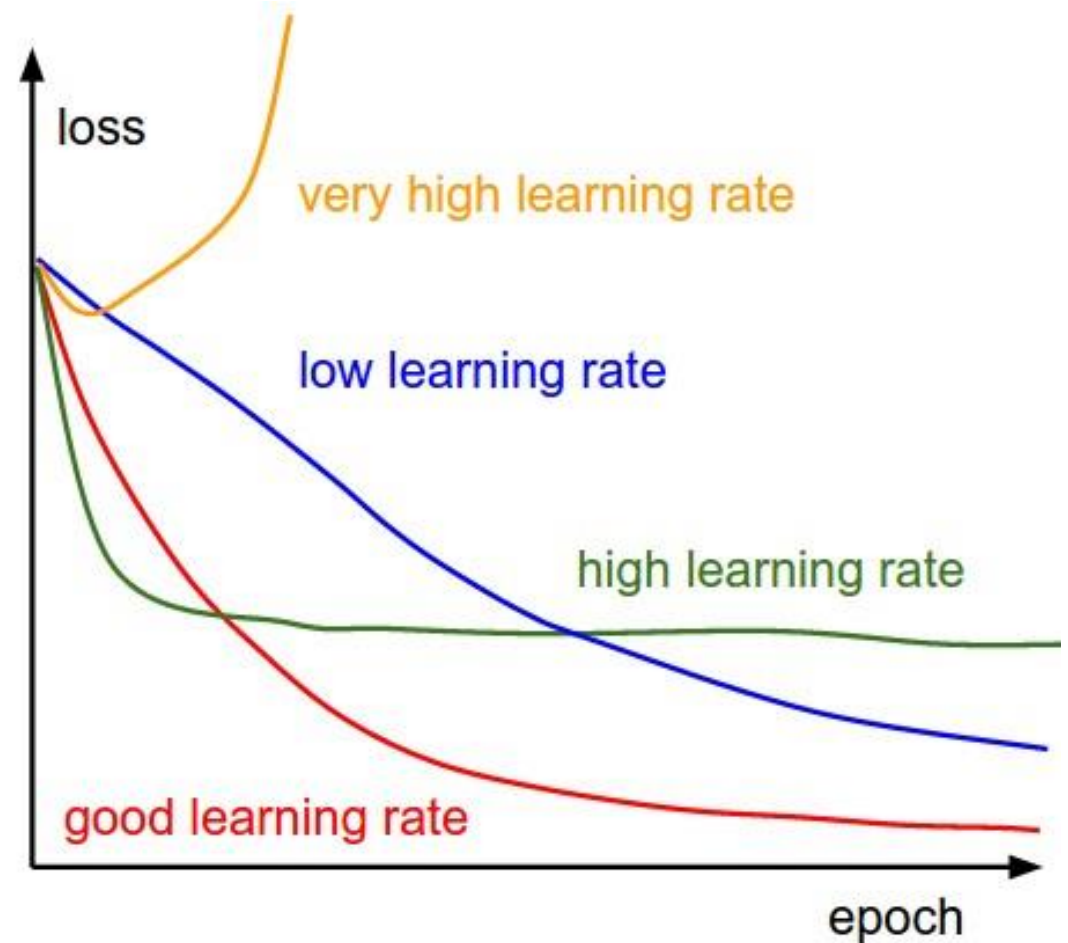


Gradient Descent in Practice: Step Size

- 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

A Simple Approach to Curve Fitting

Representation

- Fit the data using a *polynomial function*

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

– where M is the order of the polynomial

- Is higher value of M better? We'll see shortly!
- Coefficients $w_0, \dots, w_M$ 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)

Polynomial Curve Fitting

– With a single input variable x

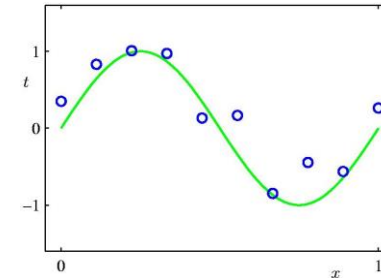
– $y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M$

M = order of polynomial,

x^j denotes x raised to power j ,

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

$$= \sum_{j=0}^M w_j x^j$$



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

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

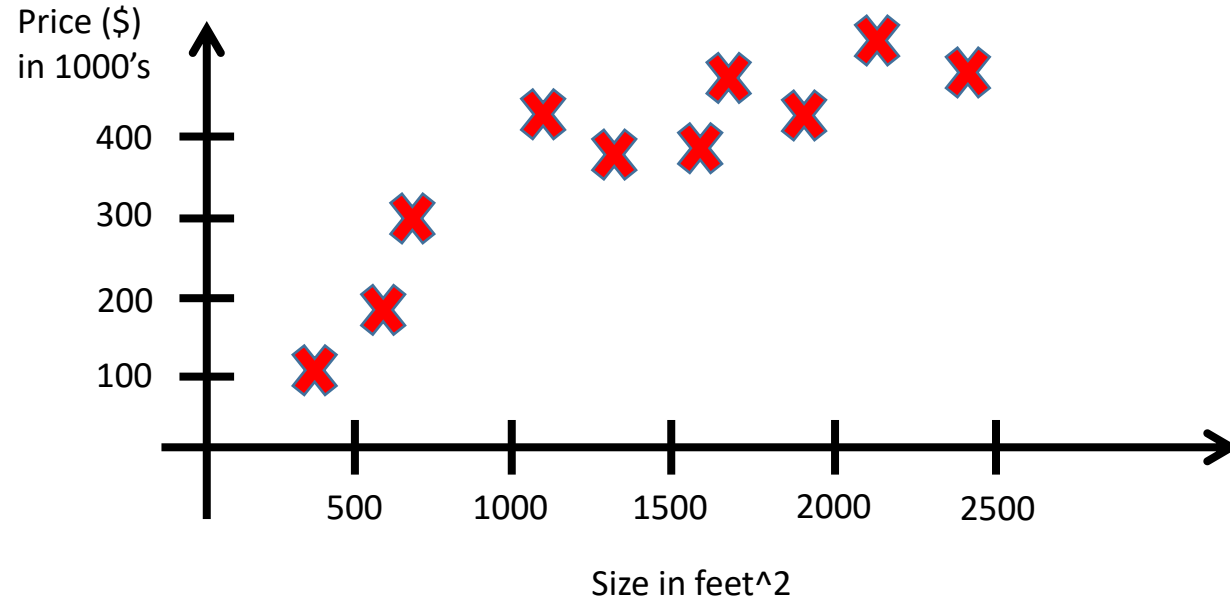
- Can be done by minimizing an error function that minimizes misfit between $y(x, \mathbf{w})$ for any given $\mathbf{w}$ and training data
- One simple choice of error function is sum of squares of error (SSE) between predictions $y(x_n, \mathbf{w})$ for each data point x_n and corresponding target 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

A Polynomial Hypothesis

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

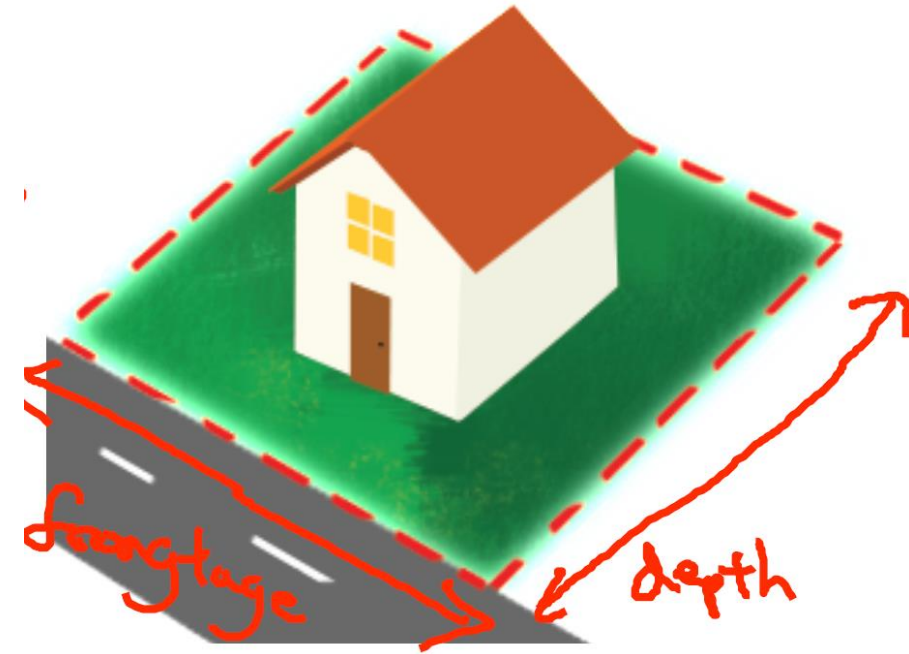


$$\begin{aligned}x_1 &= (\text{size}) \\x_2 &= (\text{size})^2 \\x_3 &= (\text{size})^3\end{aligned}$$

- $$\begin{aligned}h_{\theta}(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\&= \theta_0 + \theta_1 (\text{size}) + \theta_2 (\text{size})^2 + \theta_3 (\text{size})^3\end{aligned}$$

House Price(s) Prediction

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

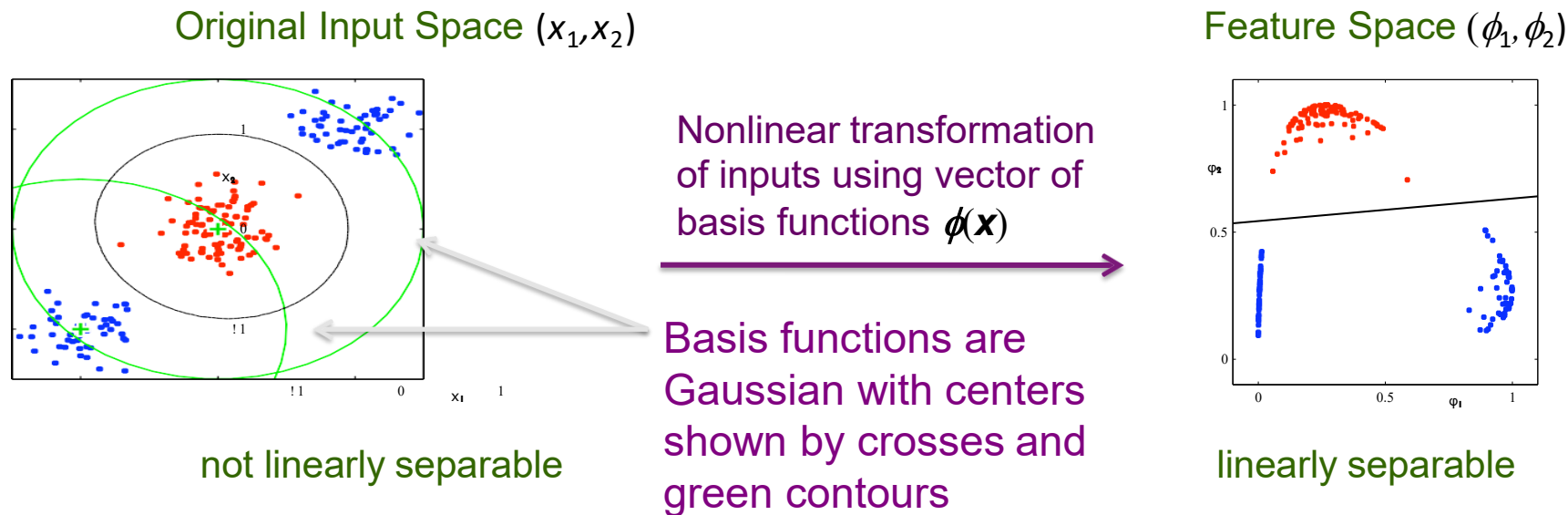


On Basis Functions

- 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

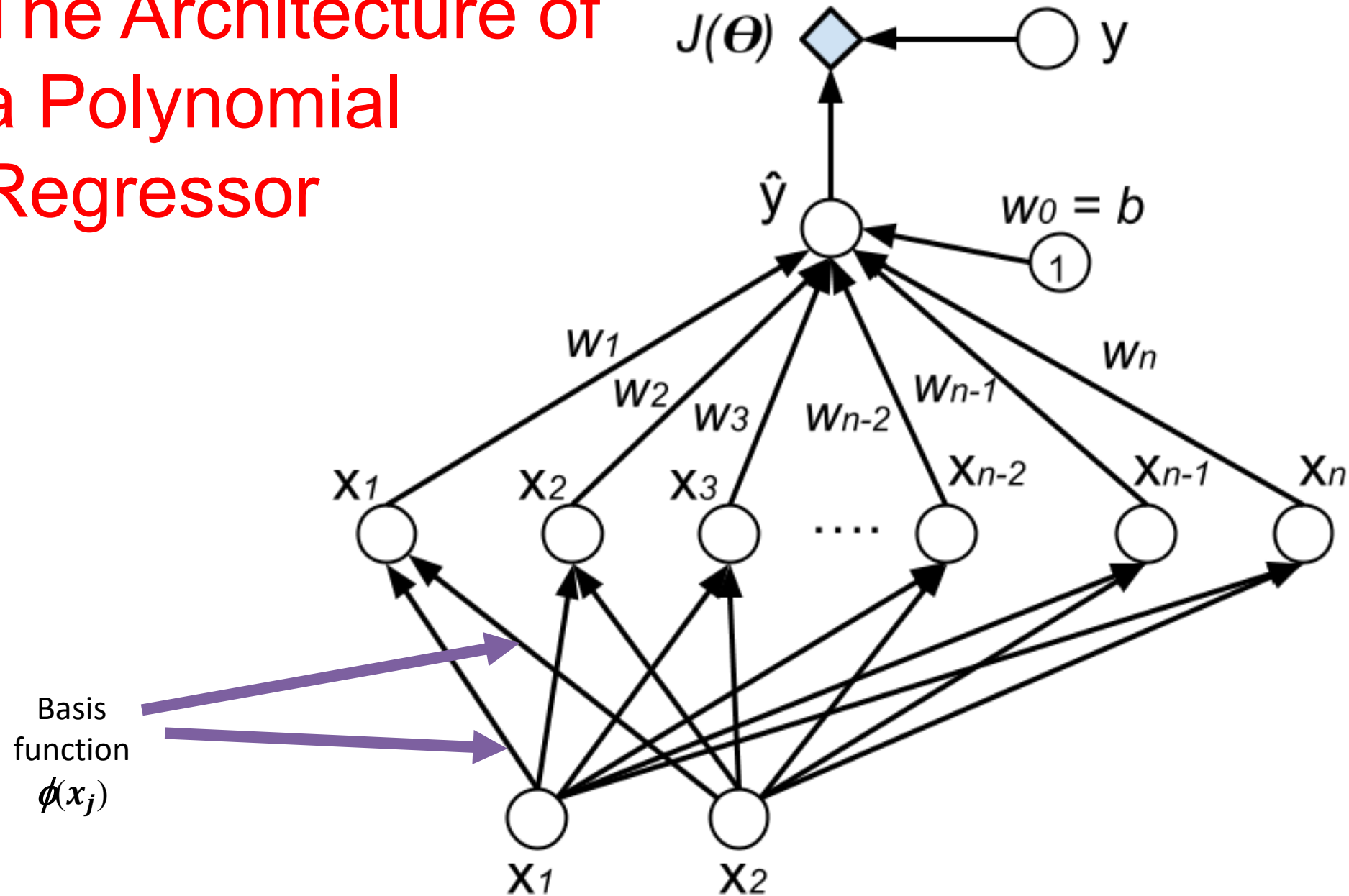
Remember, Fixed Basis Functions

Although we use linear (classification) models
Linear-separability in *feature* space *does not* imply
linear-separability in *input* space



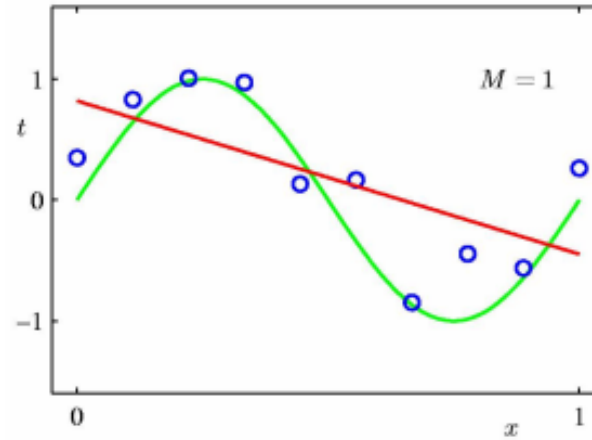
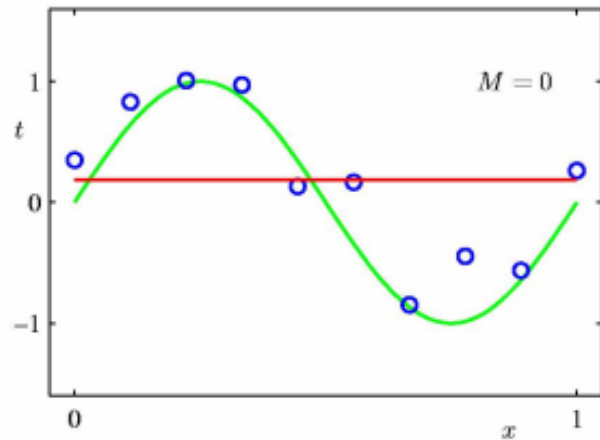
Basis functions with increased dimensionality often used

The Architecture of a Polynomial Regressor

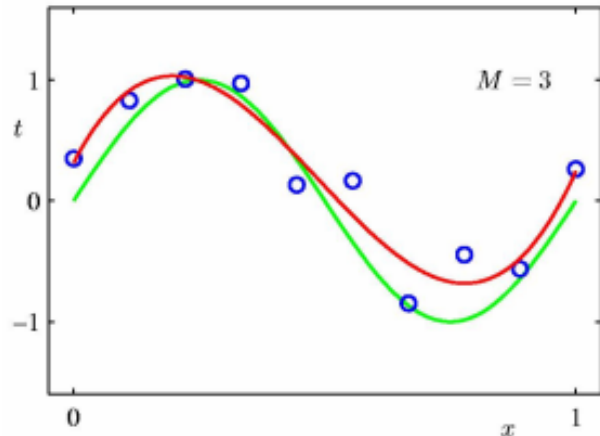


Choosing the Order of M

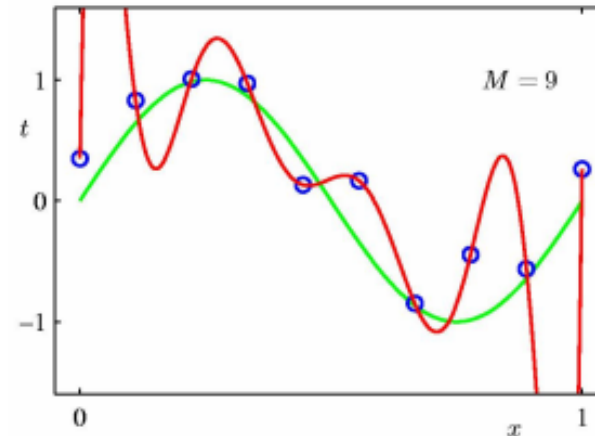
- Model Comparison or Model Selection
- Red lines are best fits with
 - $M = 0, 1, 3, 9$ and $N=10$



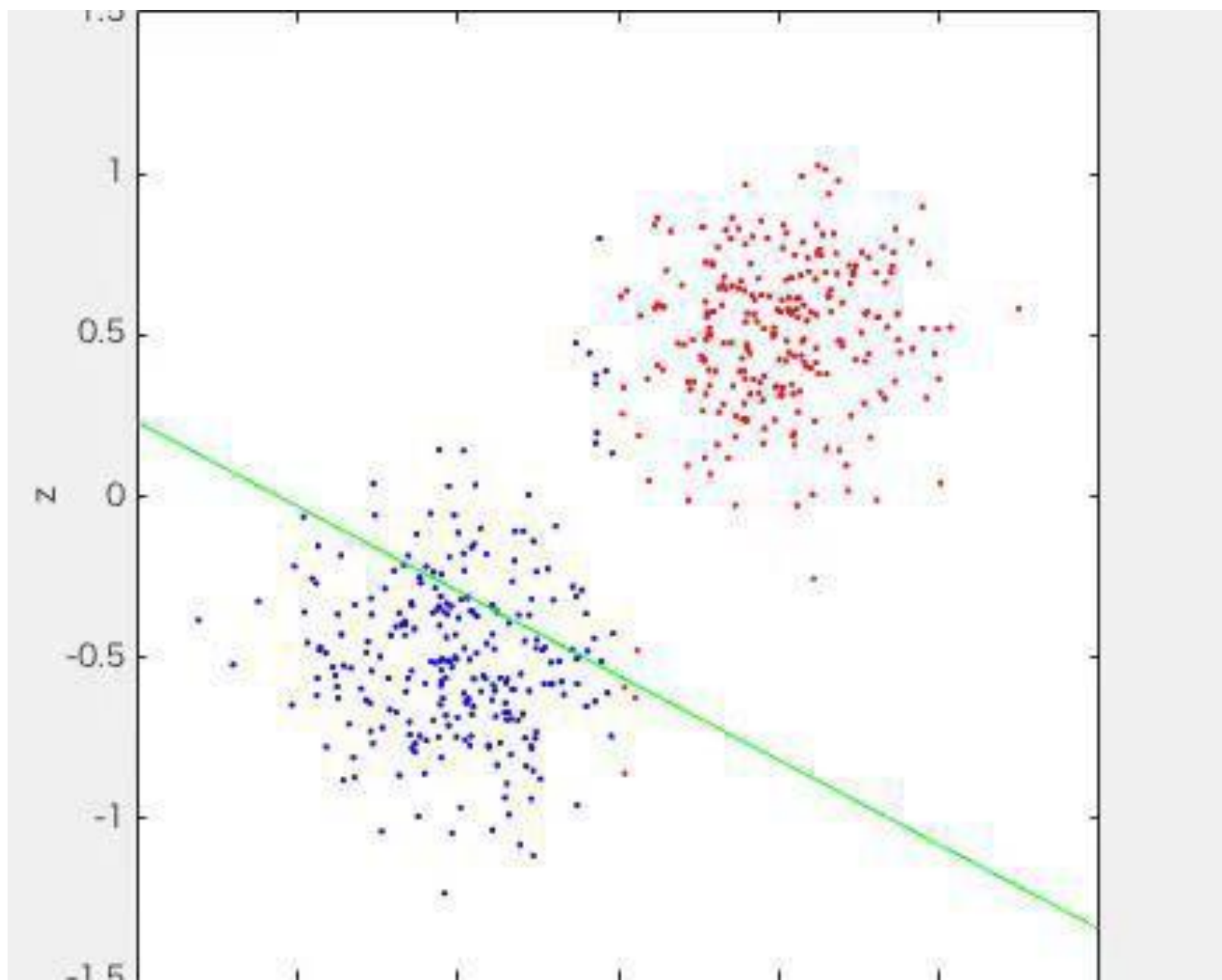
← Poor representations of $\sin(2\pi x)$

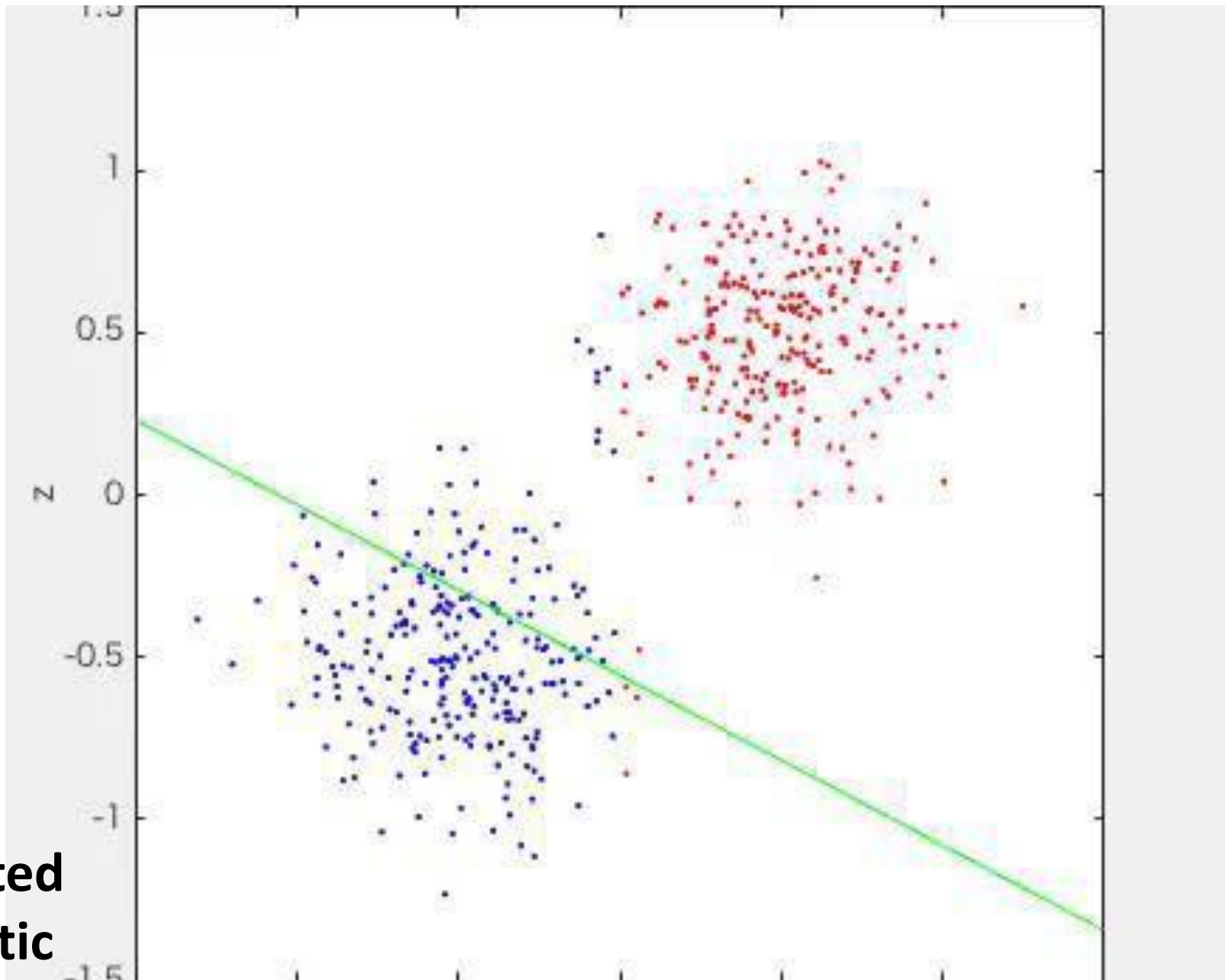


← Best Fit to $\sin(2\pi x)$



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





**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 $\alpha \in [0, \infty)$ 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:

$$w \leftarrow w - \varepsilon (\alpha w + \nabla_w J(w; X, y))$$

- Written another way, the update is

$$w \leftarrow (1 - \varepsilon \alpha) w - \varepsilon \nabla_w J(w; X, y)$$

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

Multivariate Regressor Architecture

$$f_{\Theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \leftarrow \text{Your Hypothesis!}$$

Cost:

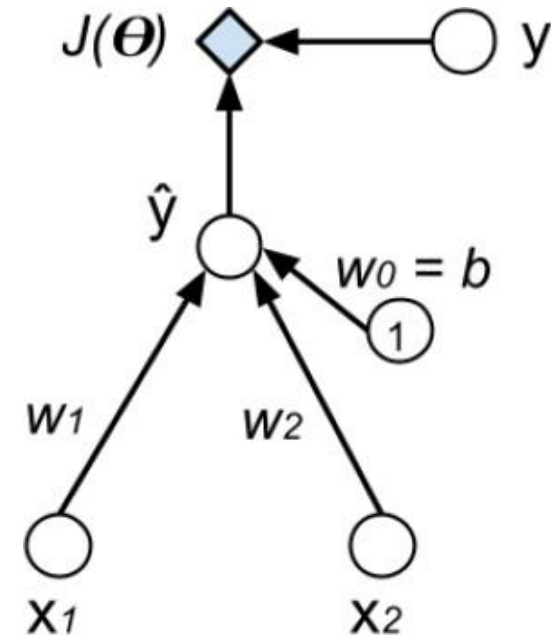
$$\mathcal{J}(\Theta) = \frac{1}{2m} \sum_{i=1}^m (f_{\Theta}(x^i) - y^i)^2 + \frac{\beta}{2m} \sum_{j=1}^n \theta_j^2$$

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:

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



Questions?

Deep robots!

Deep questions?!

