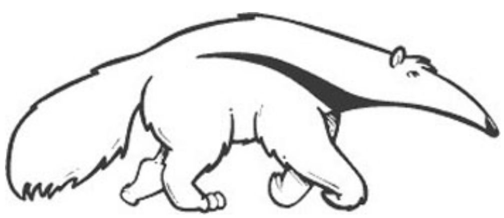
Machine Learning and Data Mining

Linear regression

Prof. Alexander Ihler

Fall 2012

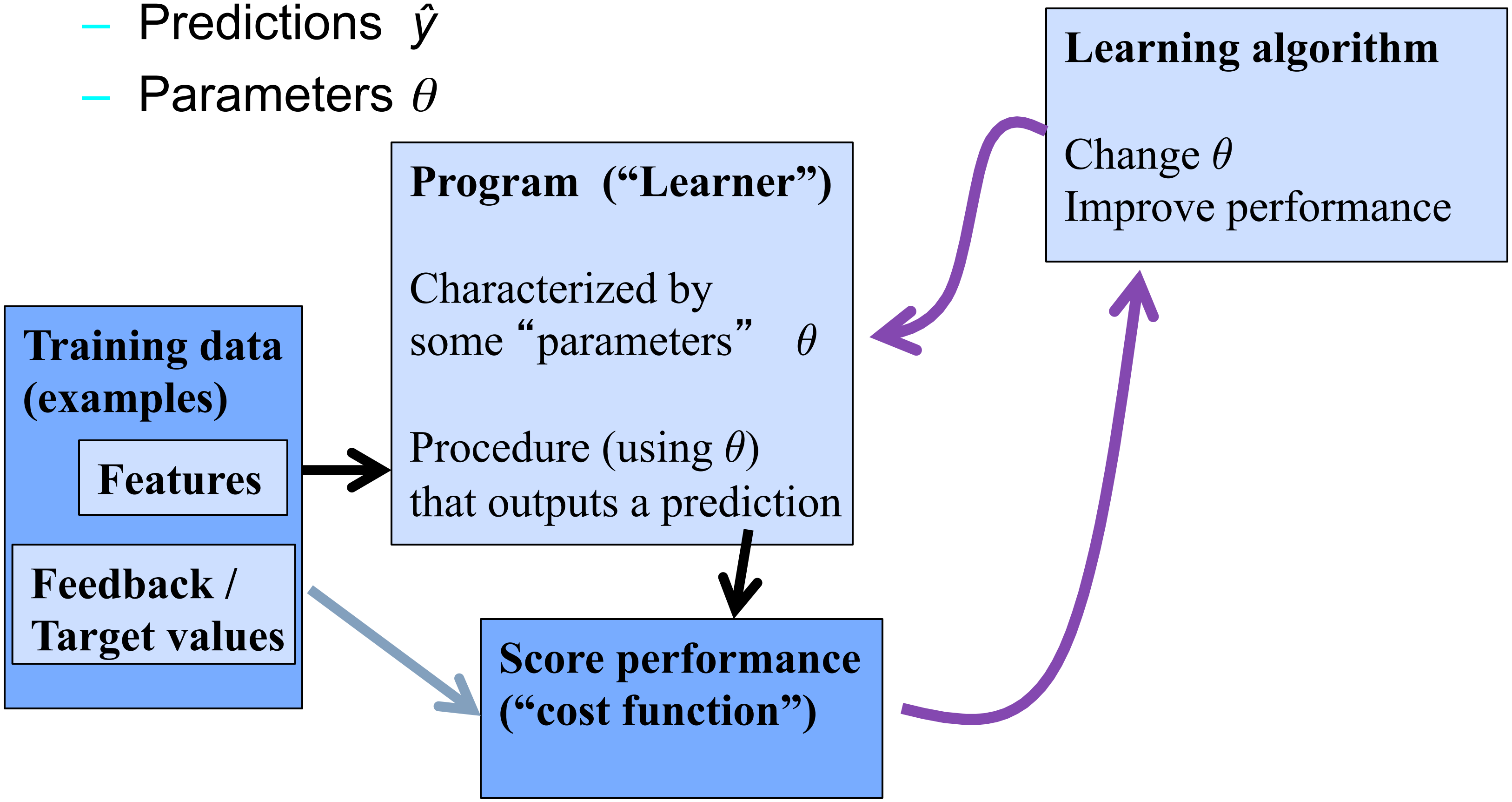


BREN:ICS
INFORMATION AND COMPUTER SCIENCES

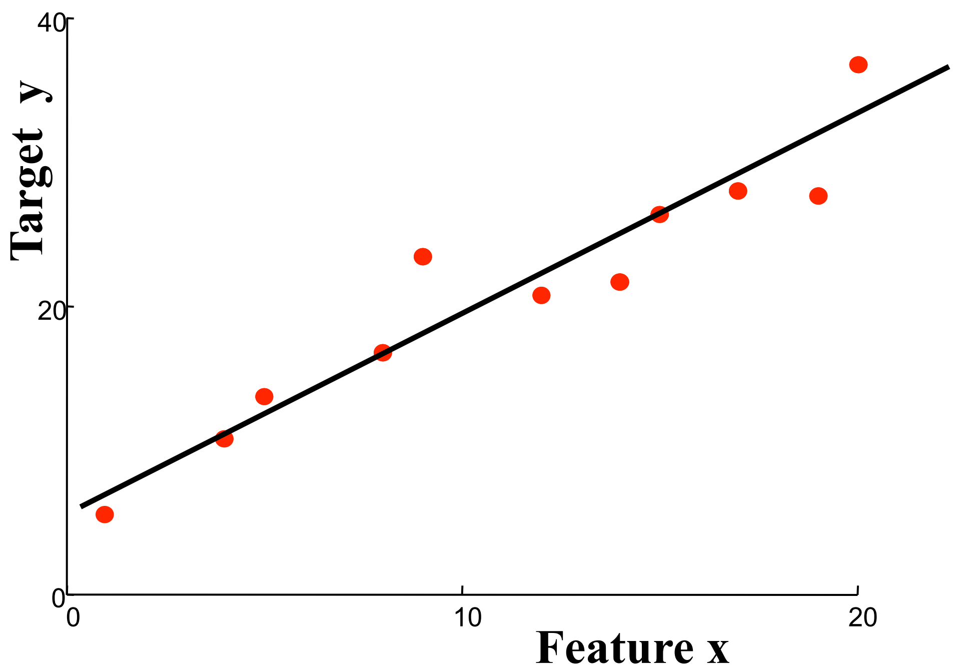
UNIVERSITY of CALIFORNIA IRVINE

Supervised learning

- Notation
  - Features x
  - Targets y
  - Predictions $\hat{y}$
  - Parameters θ



Linear regression



"Predictor":
Evaluate line:

$$r = \theta_0 + \theta_1 x_1$$

return r

- Define form of function f(x) explicitly
- Find a good f(x) within that family

Notation

$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots$$

Define "feature" x0 = 1 (constant)

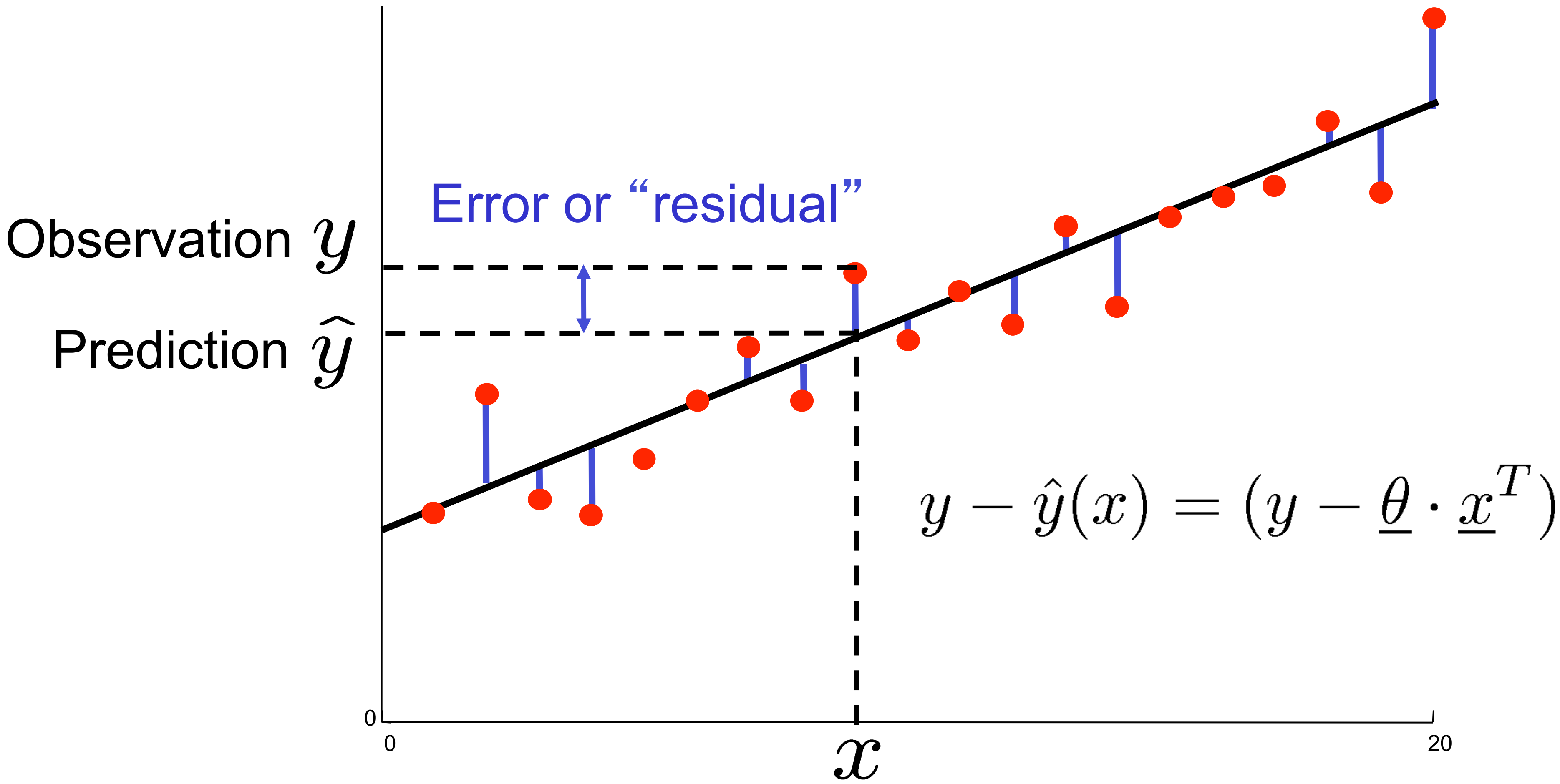
Then

$$\hat{y}(x) = \theta \, x^T$$

$$\underline{\theta} = [\theta_0, \ldots, \theta_n]$$

$$\underline{x} = [1, x_1, \ldots, x_n]$$

Measuring error



Sum of squared error

- How can we quantify the error?

$$\text{SSE, } J(\underline{\theta}) = \frac{1}{2}\sum_j (y^{(j)} - \hat{y}(x^{(j)}))^2$$
$$= \frac{1}{2}\sum_j (y - \underline{\theta} \cdot \underline{x}^T)^2$$

- Could choose something else, of course…
  - Computationally convenient (more later)
  - Measures the variance of the residuals
  - Corresponds to Gaussian model of “noise”

$$\mathcal{N}(y \;;\; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y-\mu)^2\right\}$$

SSE cost function

$$\text{SSE},\ J(\underline{\theta}) = \frac{1}{2}\sum_j (y^{(j)} - \hat{y}(x^{(j)}))^2$$

$$= \frac{1}{2}\sum_j (y - \underline{\theta} \cdot \underline{x}^T)^2$$

- Rewrite using matrix form

$$\underline{\theta} = [\theta_0, \ldots, \theta_n]$$

$$\underline{y} = \left[y^{(1)} \ldots, y^{(m)}\right]$$

$$\underline{X} = \begin{bmatrix} x_0^{(1)} & \ldots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \ldots & x_n^{(m)} \end{bmatrix}$$

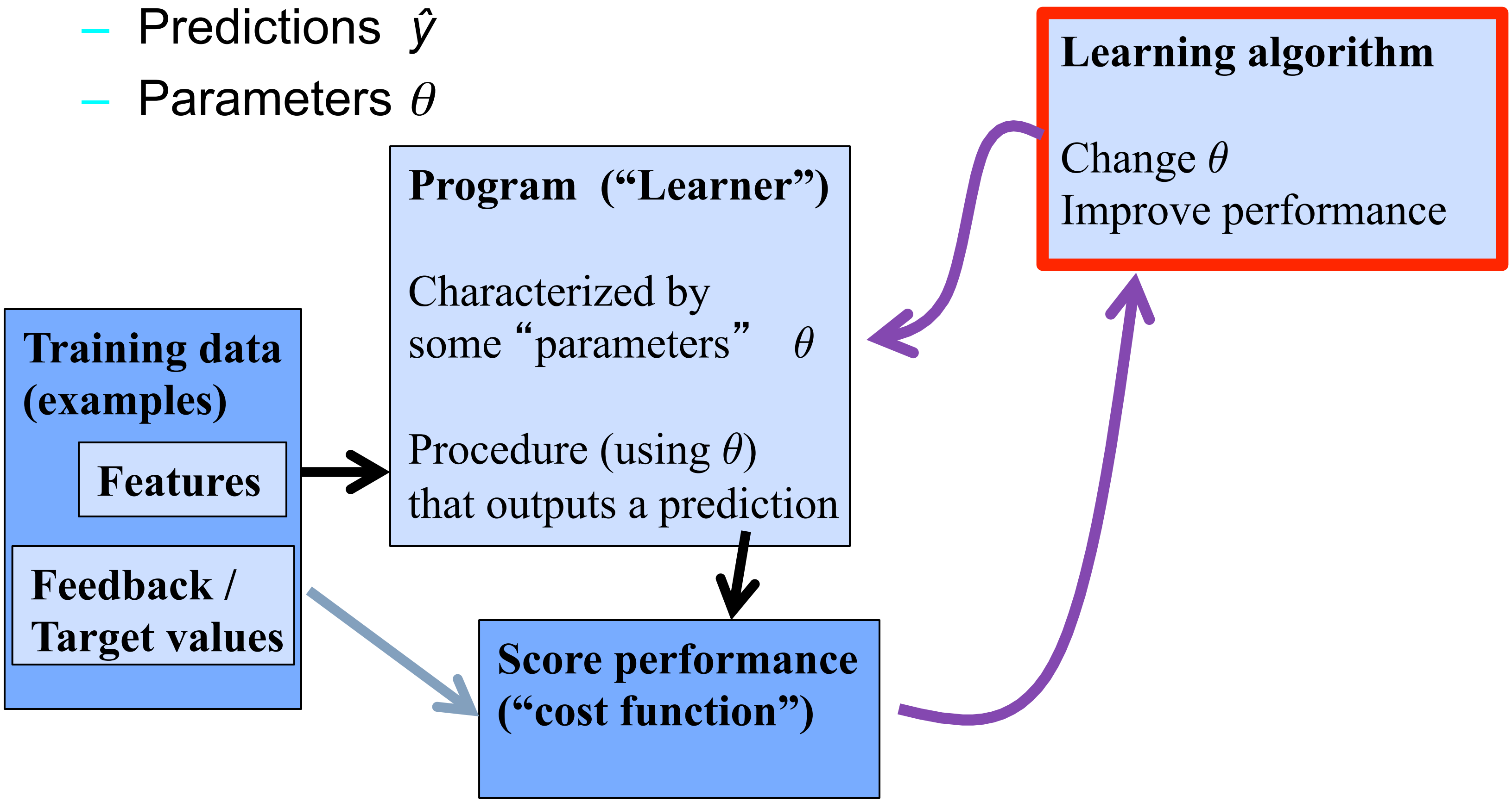
$$J(\underline{\theta}) = \frac{1}{2}(\underline{y} - \underline{\theta}\,\underline{X}^T) \cdot (\underline{y} - \underline{\theta}\,\underline{X}^T)^T$$

(Matlab)

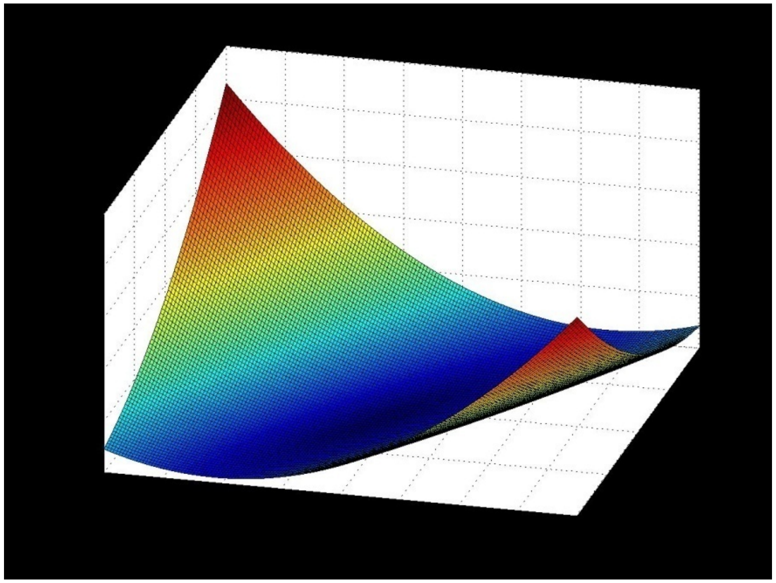
```
>> e = y - th*X';   J = .5*e*e';
```

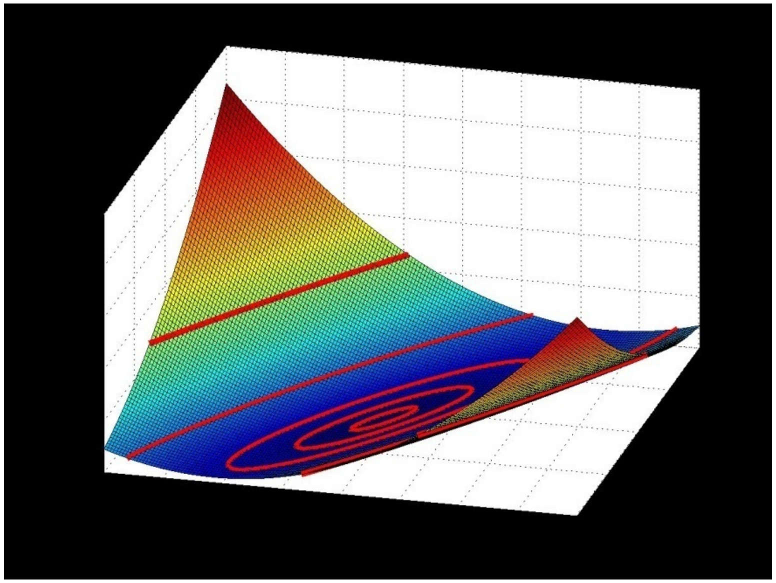
Supervised learning

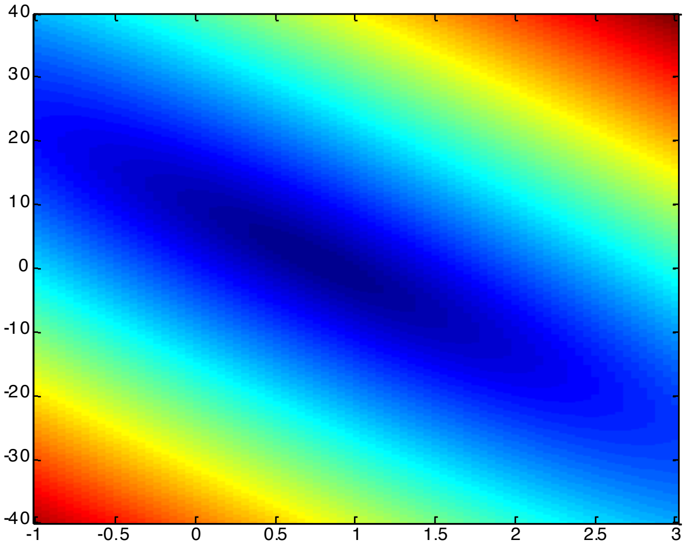
- Notation
  - Features x
  - Targets y
  - Predictions $\hat{y}$
  - Parameters θ

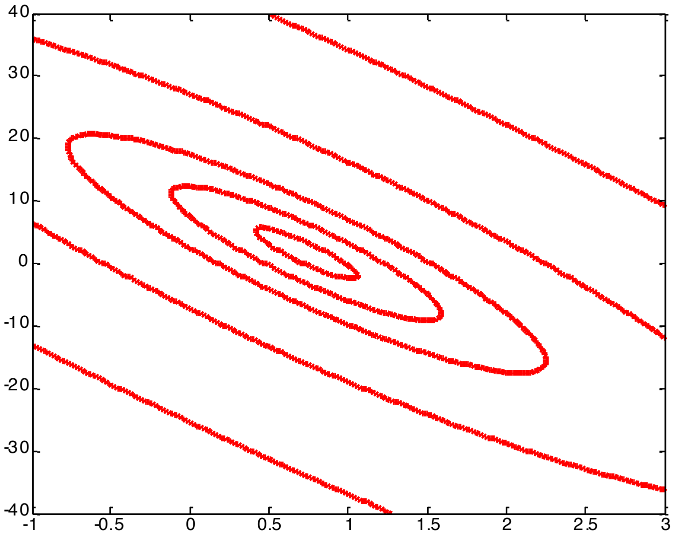


Visualizing the cost function



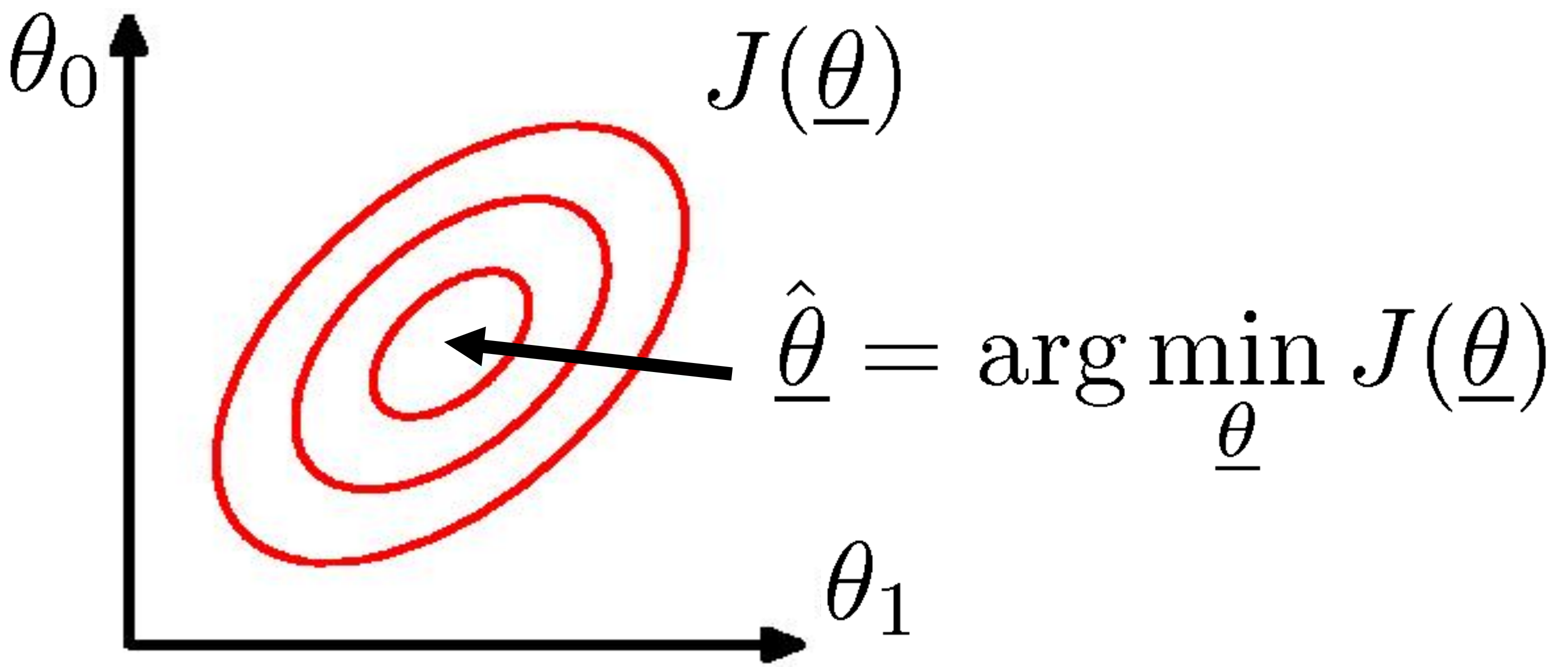




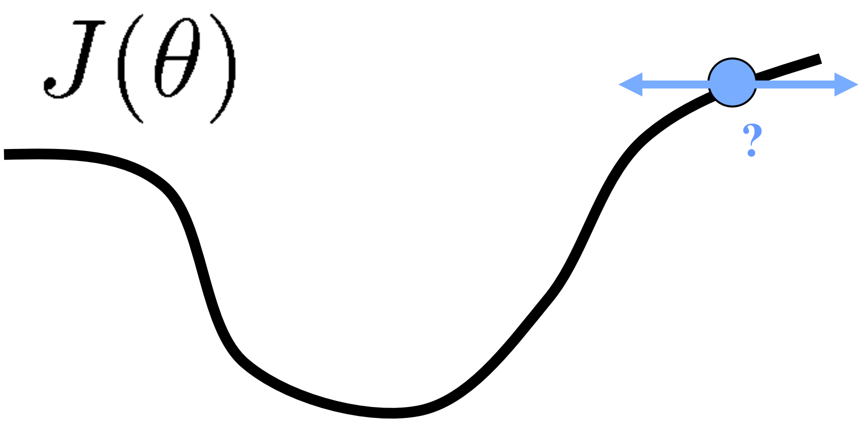


Finding good parameters

- Want to find parameters which minimize our error…
- Think of a cost "surface": error residual for that θ…

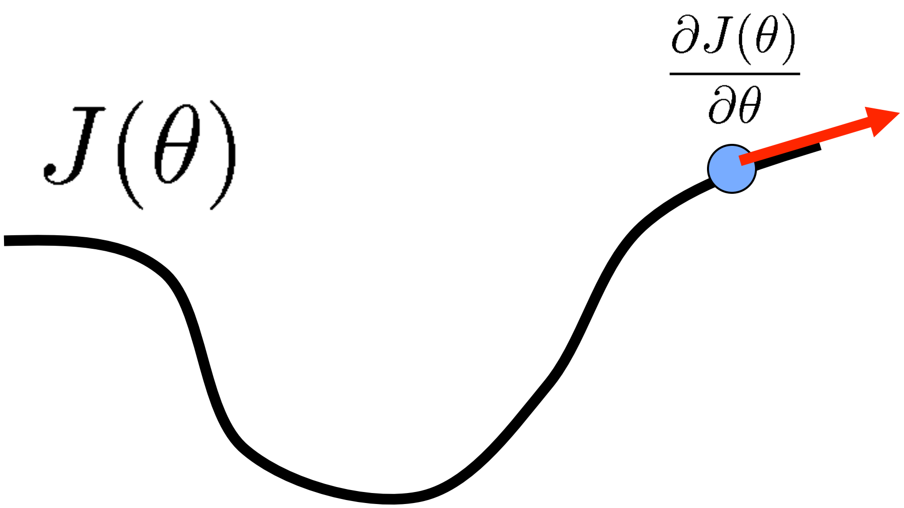


Gradient descent



- How to change θ to improve J(θ)?
- Choose a direction in which J(θ) is decreasing

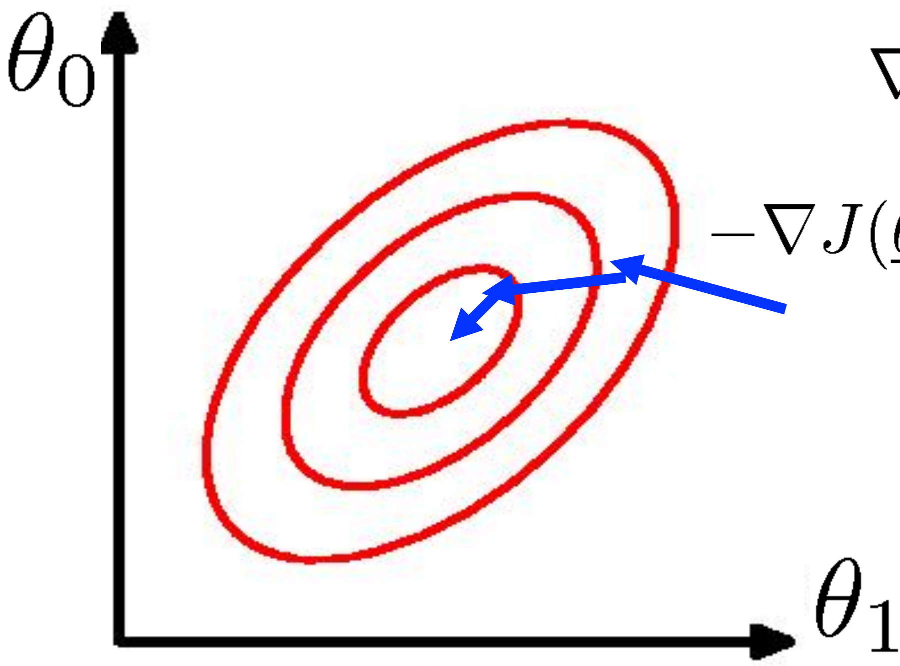
Gradient descent



- How to change θ to improve J(θ)?
- Choose a direction in which J(θ) is decreasing
- Gradient $\frac{\partial J(\theta)}{\partial \theta}$

- Positive => increasing
- Negative => decreasing

Gradient descent in more dimensions



- Gradient vector

$$\nabla J(\underline{\theta}) = \left[\frac{\partial J(\underline{\theta})}{\partial \theta_0} \quad \frac{\partial J(\underline{\theta})}{\partial \theta_1} \quad \ldots \right]$$

- Indicates direction of steepest ascent

  (negative = steepest descent)

Gradient descent

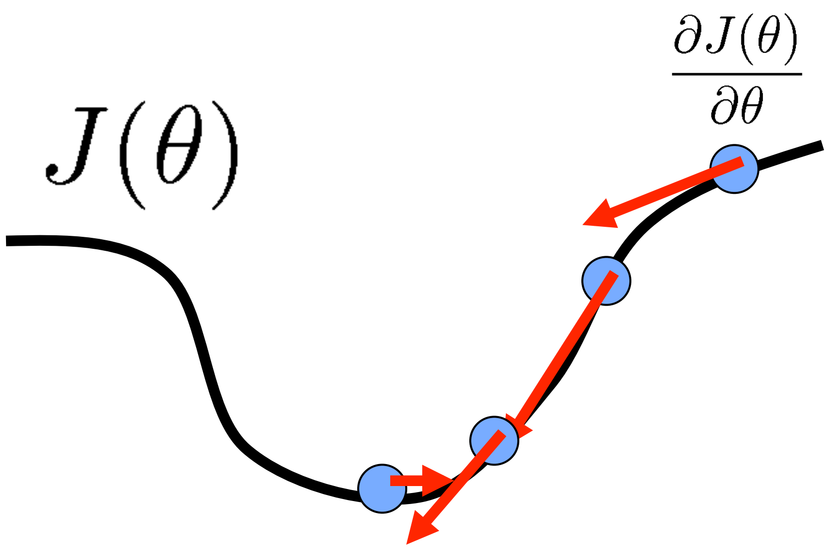
- Initialization
- Step size
  - Can change as a function of iteration
- Gradient direction
- Stopping condition

`Initialize` θ

`Do {`

$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$

`}` `while` $(\alpha||\nabla J|| > \epsilon)$



Gradient for the SSE

- SSE $$J(\underline{\theta}) = \frac{1}{2}\sum_j \left(y^{(j)} - \underline{\theta}\cdot \underline{x}^{(j)\,T}\right)^2$$

- ∇ J = ?

$$J(\underline{\theta}) = \frac{1}{2}\sum_j \left(y^{(j)} - \theta_0 \underline{x}_0^{(j)} - \theta_1 \underline{x}_1^{(j)} - \ldots\right)^2$$

$$\frac{\partial J}{\partial \theta_0} = \frac{\partial}{\partial \theta_0}\frac{1}{2}\sum_j (\; g(\theta) \;)^2$$

$$= \frac{1}{2}\sum_j \; \frac{\partial}{\partial \theta_0}(\; g(\theta) \;)^2$$

$$= \frac{1}{2}\sum_j \; 2g(\theta) \; \frac{\partial}{\partial \theta_0} g(\theta)$$

$$\frac{\partial}{\partial \theta_0} g(\theta) = \underbrace{\frac{\partial}{\partial \theta_0} y^{(j)}}_{0} - \frac{\partial}{\partial \theta_0}\theta_0 x_0^{(j)} - \underbrace{\frac{\partial}{\partial \theta_0}\theta_1 x_1^{(j)}}_{0} - \ldots$$

$$= -x_0^{(j)}$$

Gradient descent

- Initialization
- Step size
  - Can change as a function of iteration
- Gradient direction
- Stopping condition

Initialize θ

Do {

$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$

} while ($\alpha||\nabla J|| > \epsilon$)

$$J(\underline{\theta}) = \frac{1}{2}\sum_j (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)^T})^2$$

$$\nabla J(\underline{\theta}) = -\sum_j (y^{(j)} \underbrace{- \underline{\theta} \cdot \underline{x}}_{\text{Error magnitude \& direction for datum j}}{}^{(j)^T}) \cdot [\underbrace{x_0^{(j)} x_1^{(j)}}_{\text{Sensitivity to each } \theta_i} \ldots]$$

Error magnitude & direction for datum j

Sensitivity to each θ_i

Derivative of SSE

$$\nabla J(\underline{\theta}) = -\sum_j (\underbrace{y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)}}_{\text{Error magnitude \& direction for datum j}}{}^{T}) \cdot [\underbrace{x_0^{(j)} \, x_1^{(j)}}_{\text{Sensitivity to each } \theta_i} \ldots]$$

- Rewrite using matrix form

$$\underline{\theta} = [\theta_0, \ldots, \theta_n]$$

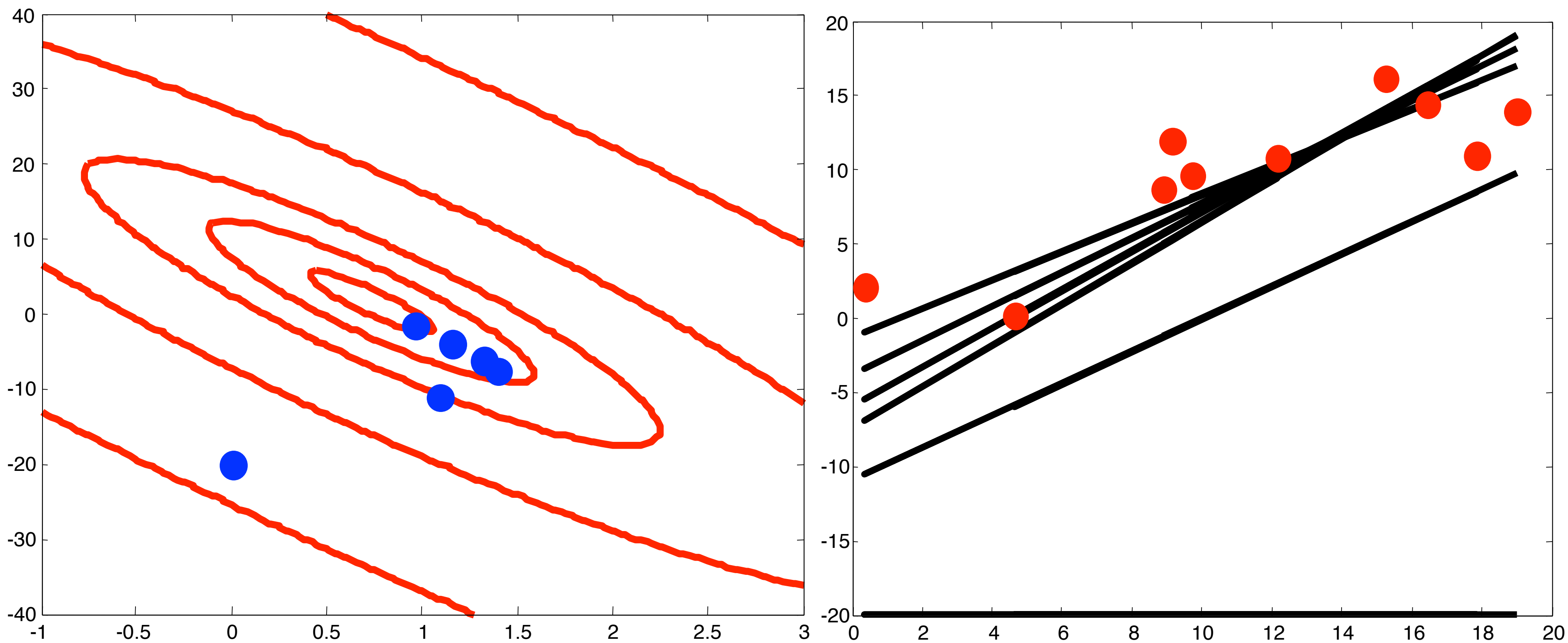
$$\underline{y} = \left[y^{(1)} \ldots, y^{(m)} \right]$$

$$\underline{X} = \begin{bmatrix} x_0^{(1)} & \ldots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \ldots & x_n^{(m)} \end{bmatrix}$$

$$\nabla J(\underline{\theta}) = (\underline{y} - \underline{\theta} \underline{X}^T) \cdot \underline{X}$$

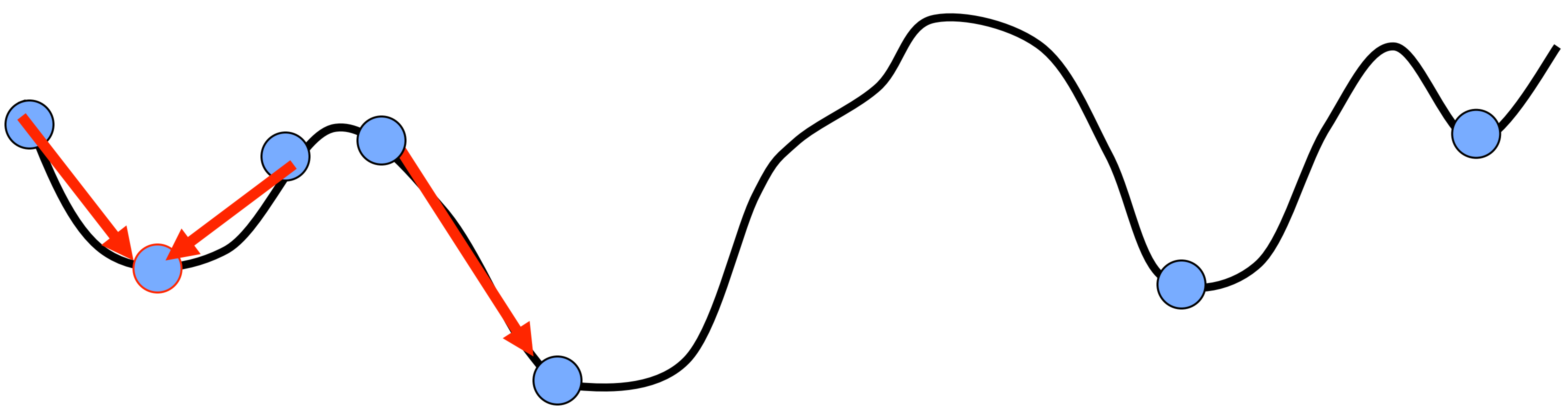
(Matlab) `>> e = y – th*X'; DJ = e*X; th=th – al*DJ;`

Gradient descent on cost function



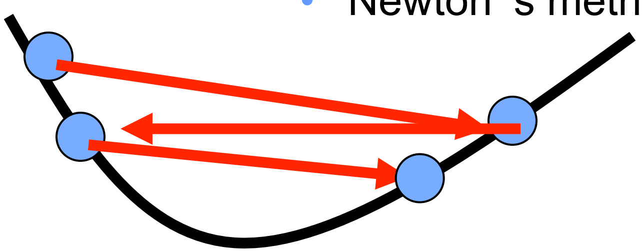
Comments on gradient descent

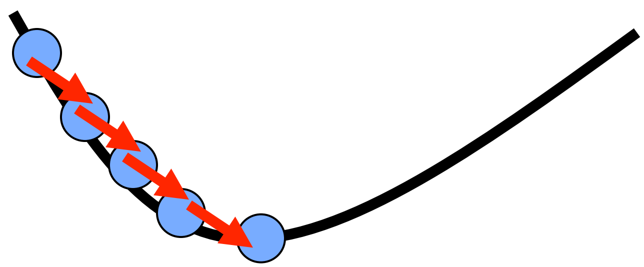
- Very general algorithm
  - we'll see it many times
- Local minima
  - Sensitive to starting point



Comments on gradient descent

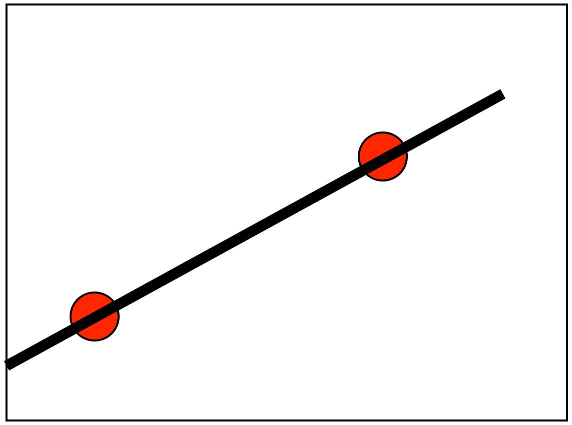
- Very general algorithm
  - we'll see it many times
- Local minima
  - Sensitive to starting point
- Step size
  - Too large? Too small? Automatic ways to choose?
  - May want step size to decrease with iteration
  - Common choices:
    - Fixed
    - Linear: C/(iteration)
    - Newton's method (we'll return to this…)





SSE Minimum

$$\nabla J(\underline{\theta}) = (\underline{y} - \underline{\theta X}^T) \cdot \underline{X} \quad = \quad \underline{0}$$



- Reordering, we have

$$\underline{y}\,\underline{X} - \underline{\theta X}^T \cdot \underline{X} \quad = \quad \underline{0}$$

$$\underline{y}\,\underline{X} = \underline{\theta X}^T \cdot \underline{X}$$

$$\underline{\theta} \quad = \quad \underline{y}\,\underline{X}(\underline{X}^T\,\underline{X})^{-1}$$

- X $(X^T X)^{-1}$ is called the "pseudo-inverse"

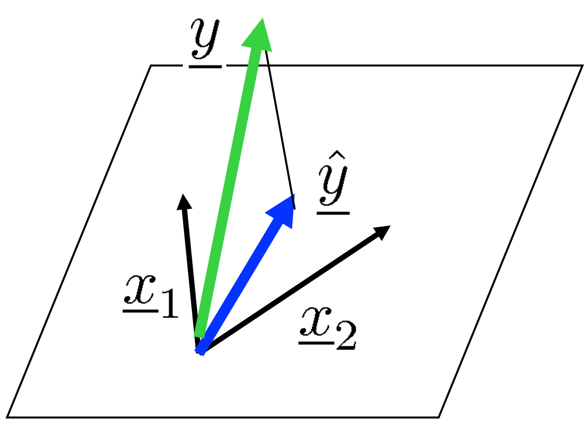
$$\underline{y} \approx \underline{\hat{y}} = \theta\,\underline{X}^T \qquad\qquad \underline{\hat{\theta}} = \underline{y} \cdot \mathrm{inv}(\underline{X}^T)$$

- If X^T is square and independent, these are the same
- If overdetermined, pseudo-inverse gives MSE estimate

Normal equations

$$\nabla J(\underline{\theta}) = 0 \quad \Rightarrow \quad (\underline{y} - \underline{\theta X}^T) \cdot \underline{X} \quad = \quad \underline{0}$$

- Interpretation:
  - (y - θX) = (y - yhat) is the vector of errors in each example
  - X are the features we have to work with for each example
  - Dot product = 0: orthogonal



$$\underline{y} = [y^{(1)} \dots y^{(m)})]$$
$$\underline{x}_i = [x_i^{(1)} \dots x_i^{(m)})]$$

Matlab SSE

- This is easy to solve in Matlab…

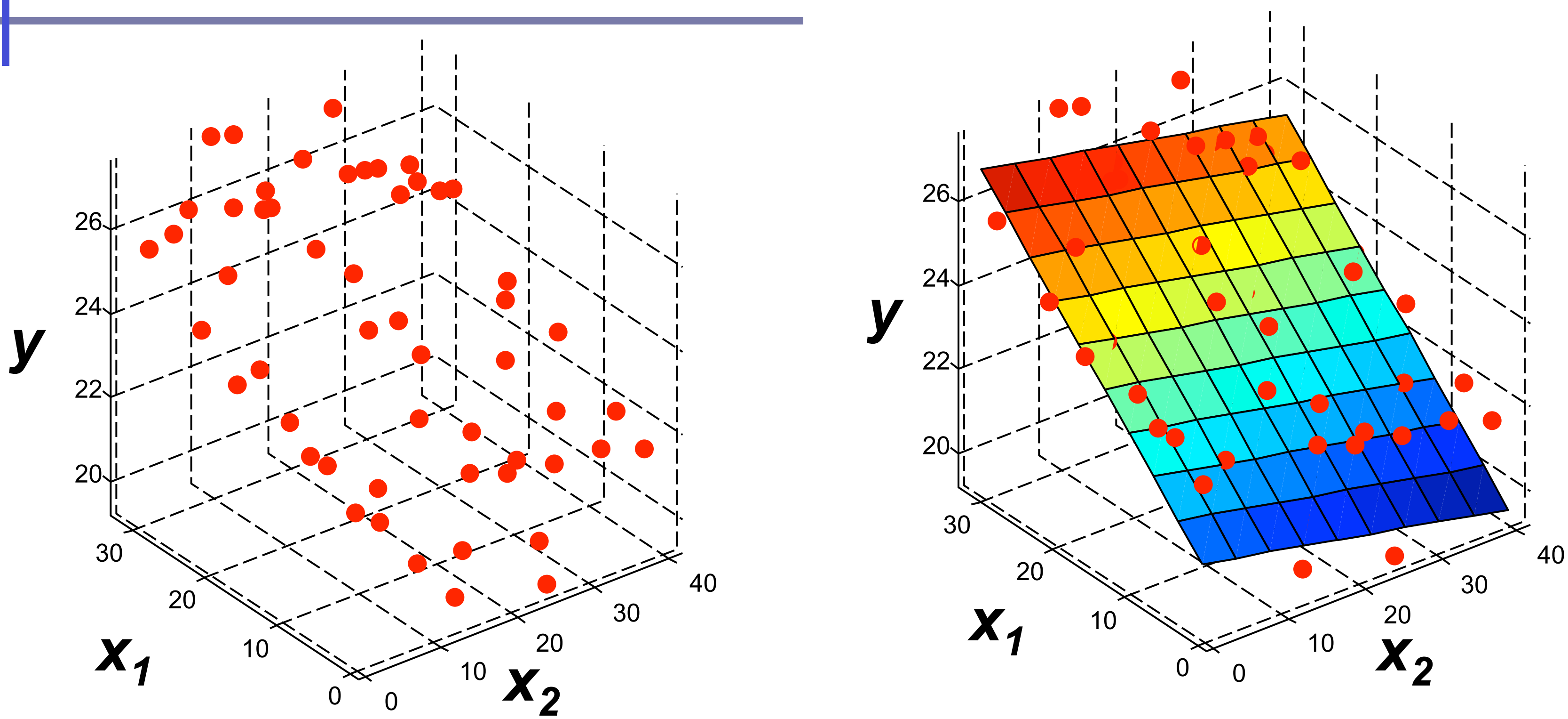
$$\underline{\theta} \quad = \quad \underline{y}\,\underline{X}(\underline{X}^T\,\underline{X})^{-1}$$

```
%   y = [y1 … ym]
%   X = [x1_0 … x1_m ; x2_0 … x2_m ; …]


% Solution 1: "manual"
    th = y * X * inv(X' * X);


% Solution 2: "mrdivide"
    th = y / X';        % th*X' = y  =>  th = y/X'
```

More dimensions?



$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

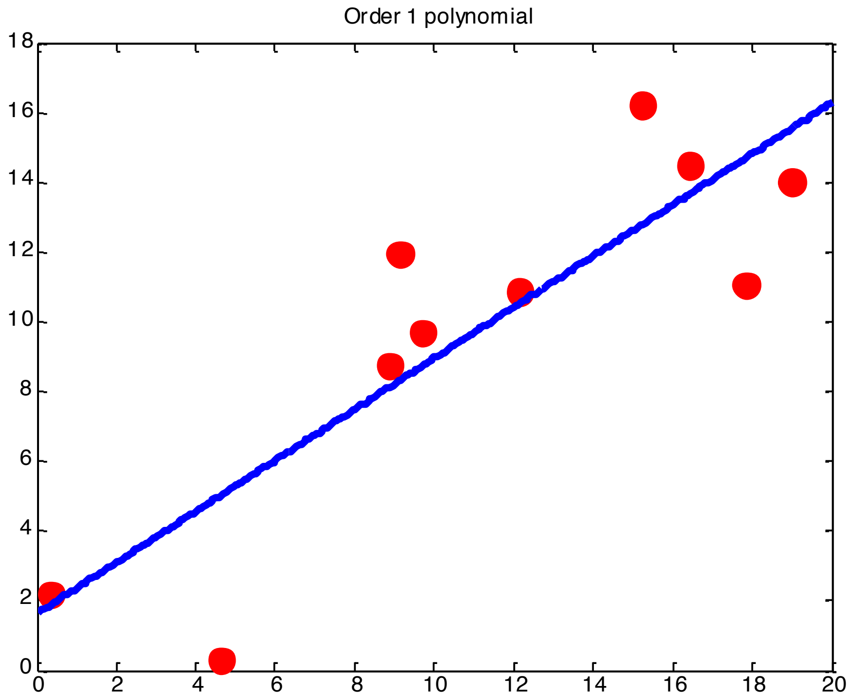
$$\underline{\theta} = [\theta_0 \ \ \theta_1 \ \ \theta_2]$$

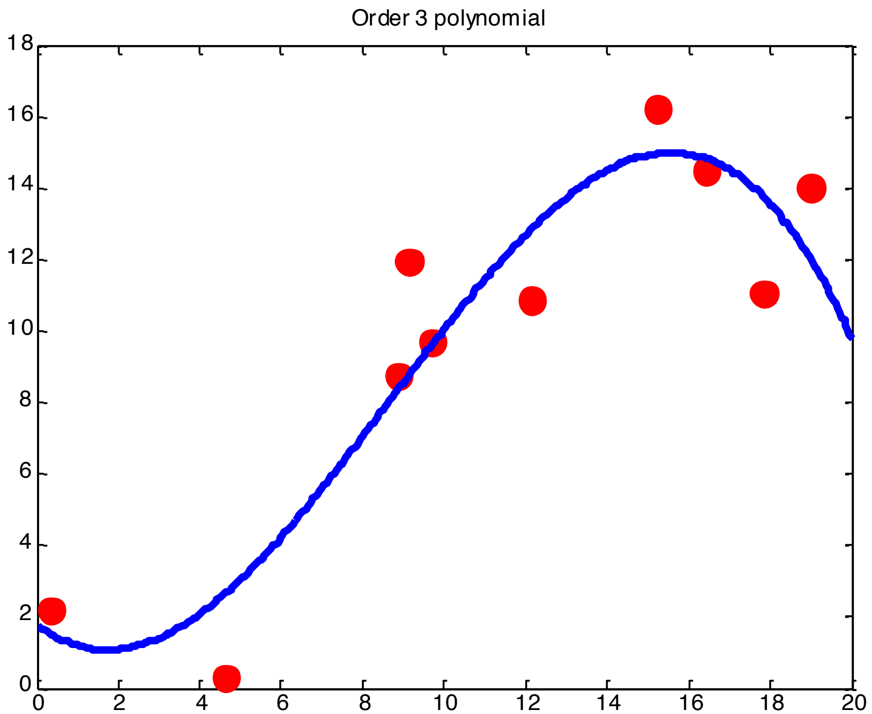
$$\hat{y}(x) = \underline{\theta} \cdot \underline{x}^T$$

$$\underline{x} = [1 \ \ x_1 \ \ x_2]$$

Nonlinear functions

- What if our hypotheses are not lines?
  - Ex: higher-order polynomials





Nonlinear functions

- Consider the polynomial in x:

$$\hat{y}(x) = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \theta_3 x^3$$

- This function is still linear in theta
  – Only nonlinear in x…

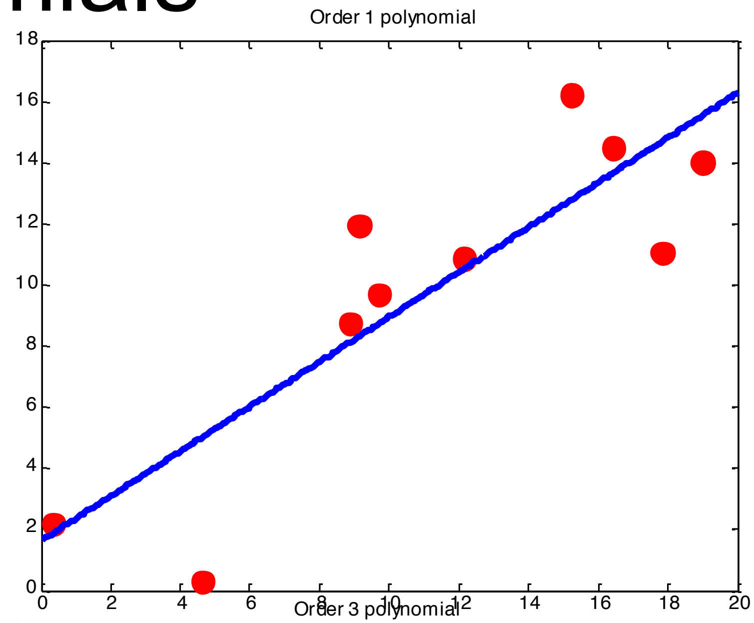
- Recall defining $x_0 = 1$
  – Let's define $x_p = x^p$
  – $x_0 = x^0 = 1$
  – $x_1 = x^1 = x$
  – $x_2 = x^2$
  – …

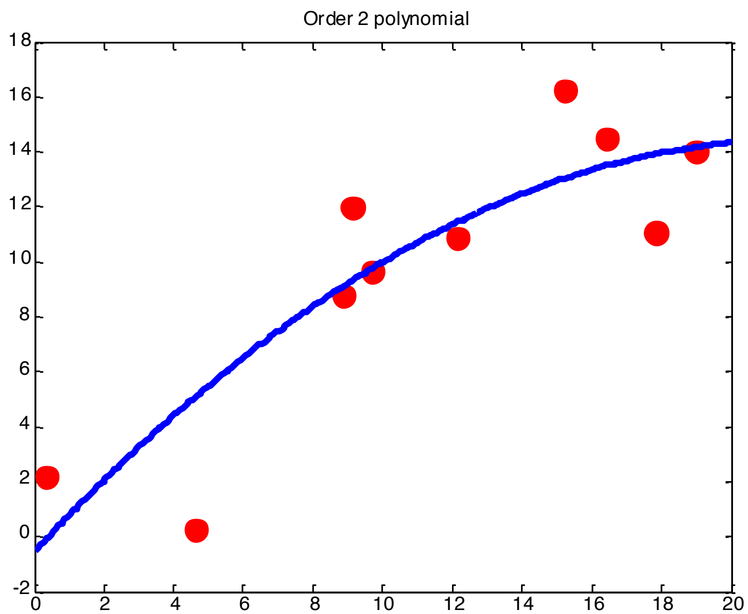
$$\hat{y}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

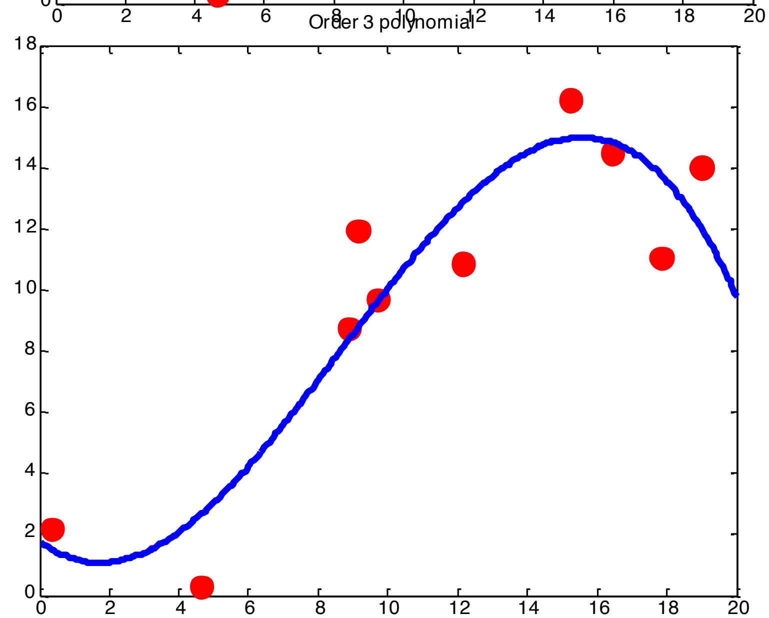
Exactly the same form as before!

Higher-order polynomials

- Fit in the same way
- More "features"







Features

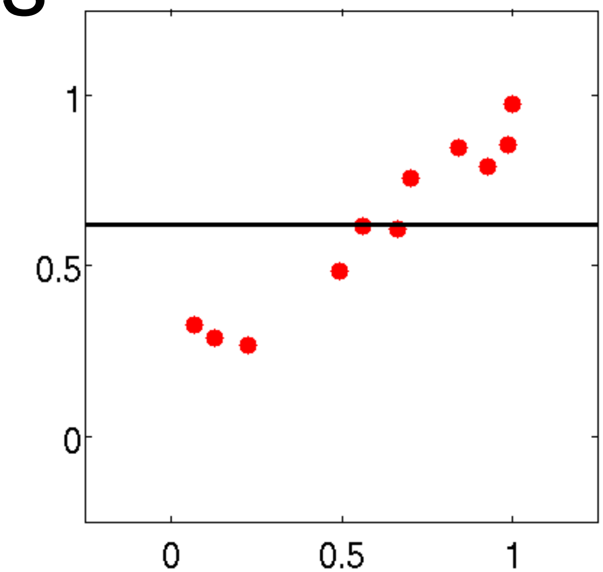
- In general, can use any features we think are useful

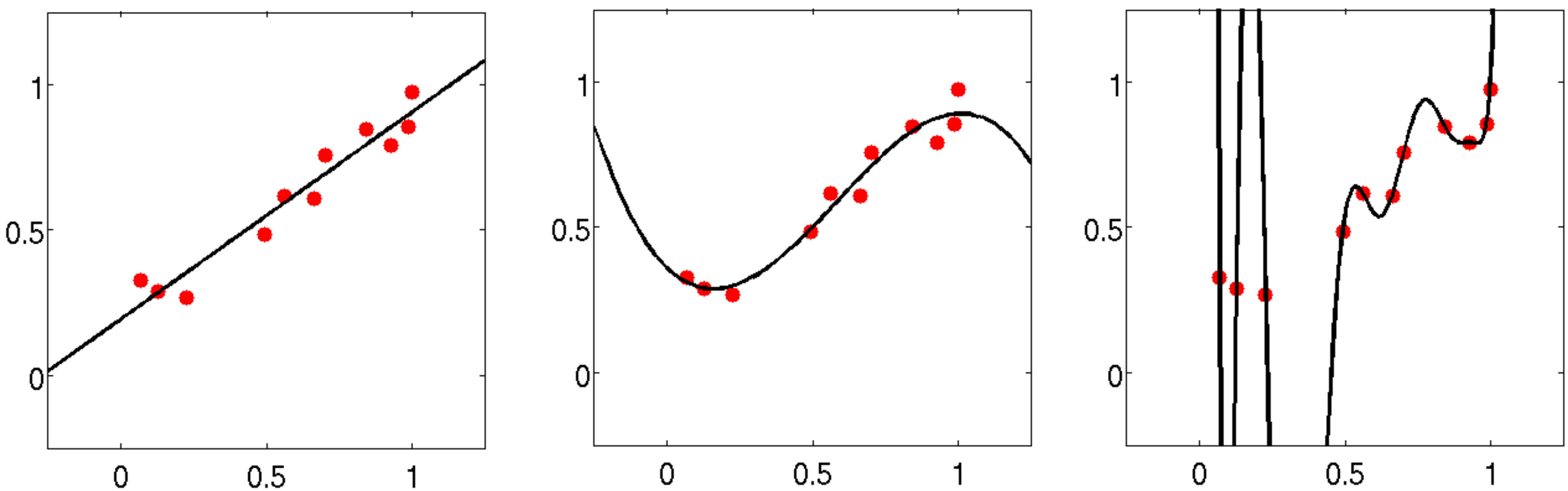
- Other information about the problem
  - Sq. footage, location, age, …
- Polynomial functions
  - Features $[1, x, x^2, x^3, \ldots]$
- Other functions
  - $1/x$, $\mathrm{sqrt}(x)$, $x_1 * x_2$, …

- "Linear regression" = linear in the parameters
  - Features we can make as complex as we want!

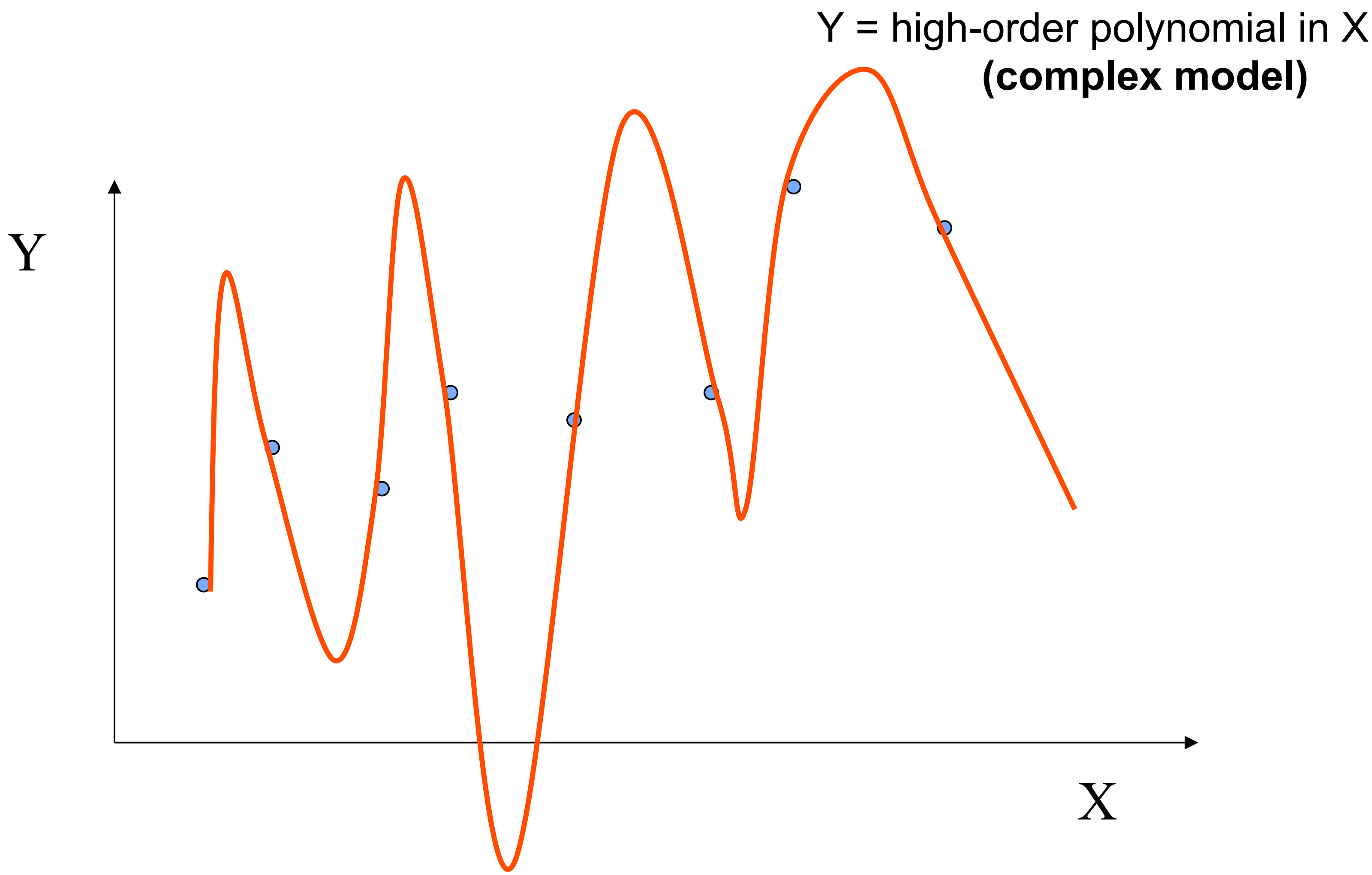
Higher-order polynomials

- Are more features better?
- “Nested” hypotheses
  - 2^{nd} order more general than 1^{st},
  - 3^{rd} order “ “ than 2^{nd}, …
- Fits the observed data better



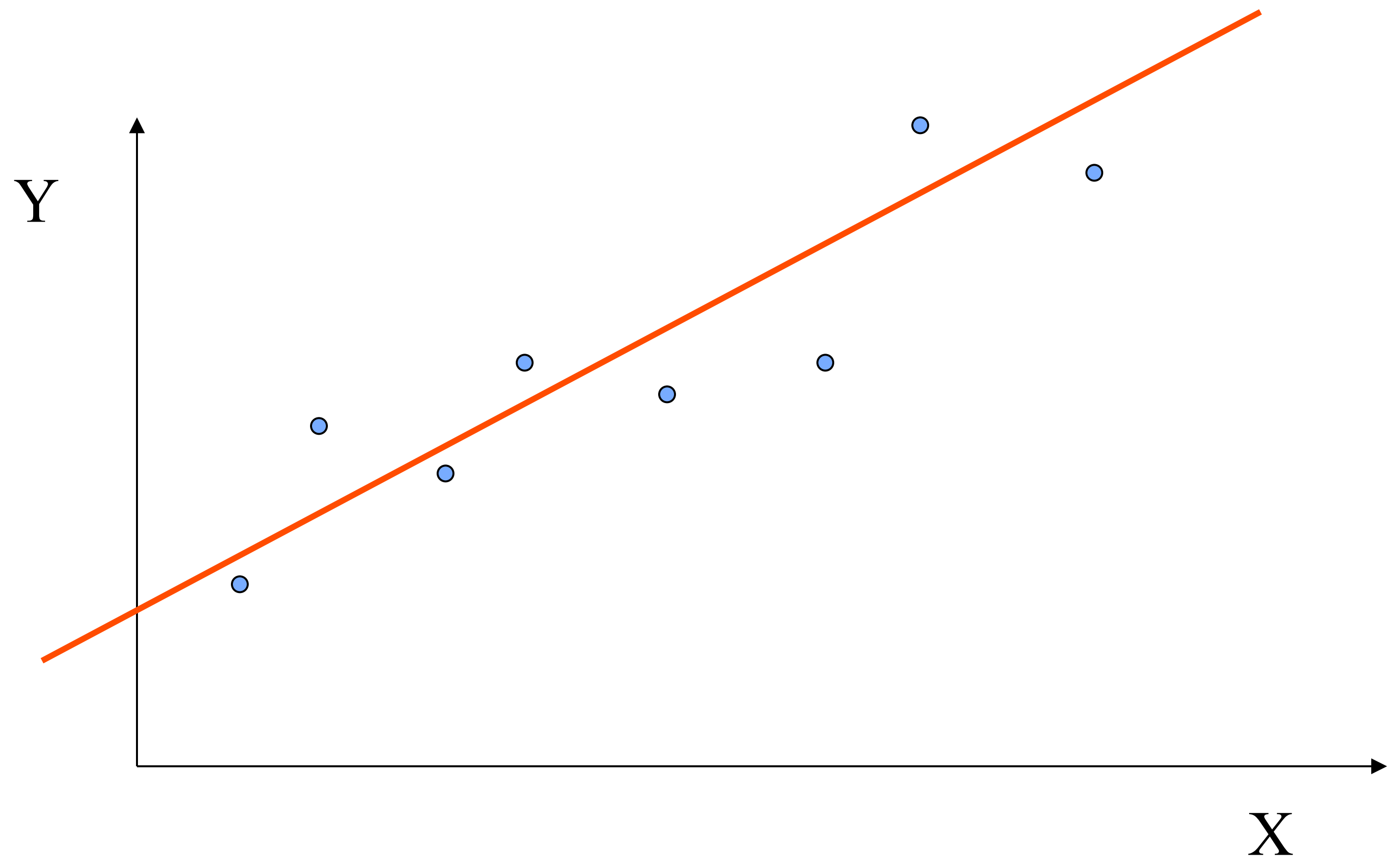


Overfitting and complexity



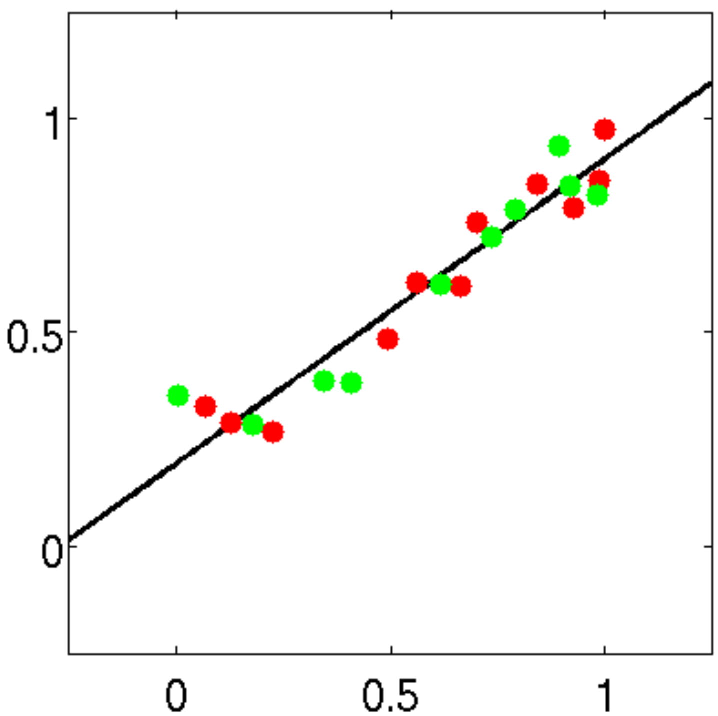
Overfitting and complexity

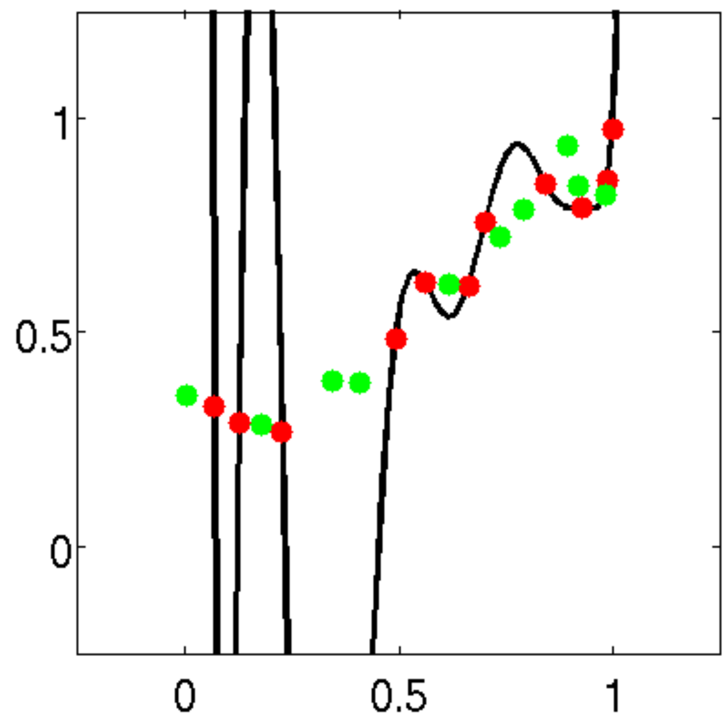
Simple model: Y= aX + b + e



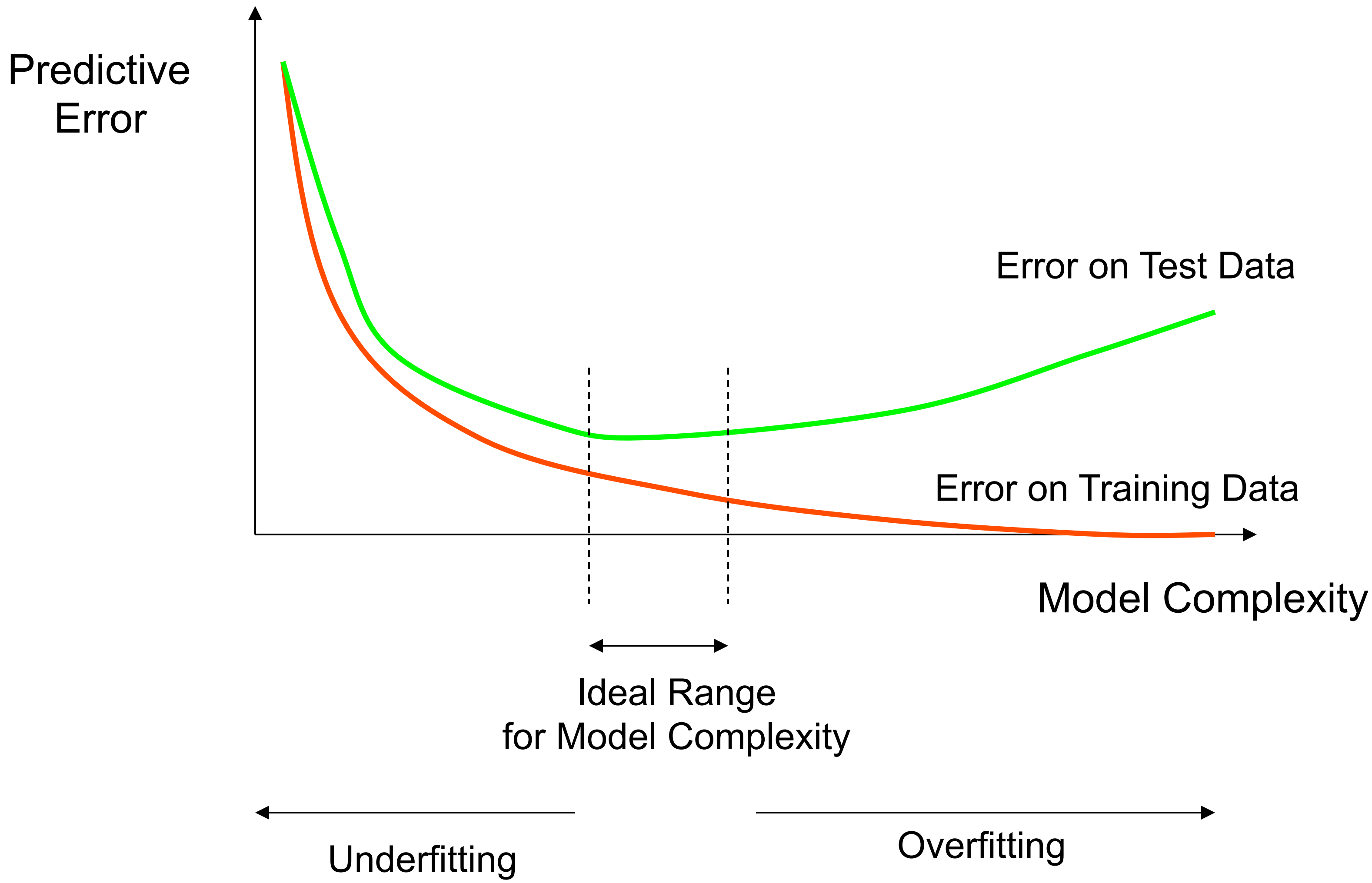
Test data

- After training the model
- Go out and get more data from the world
  - New observations (x,y)
- How well does our model perform?

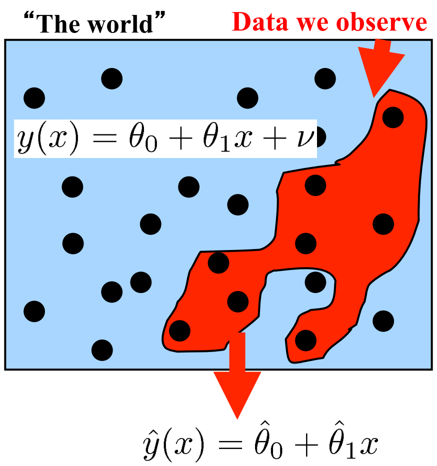




How Overfitting affects Prediction

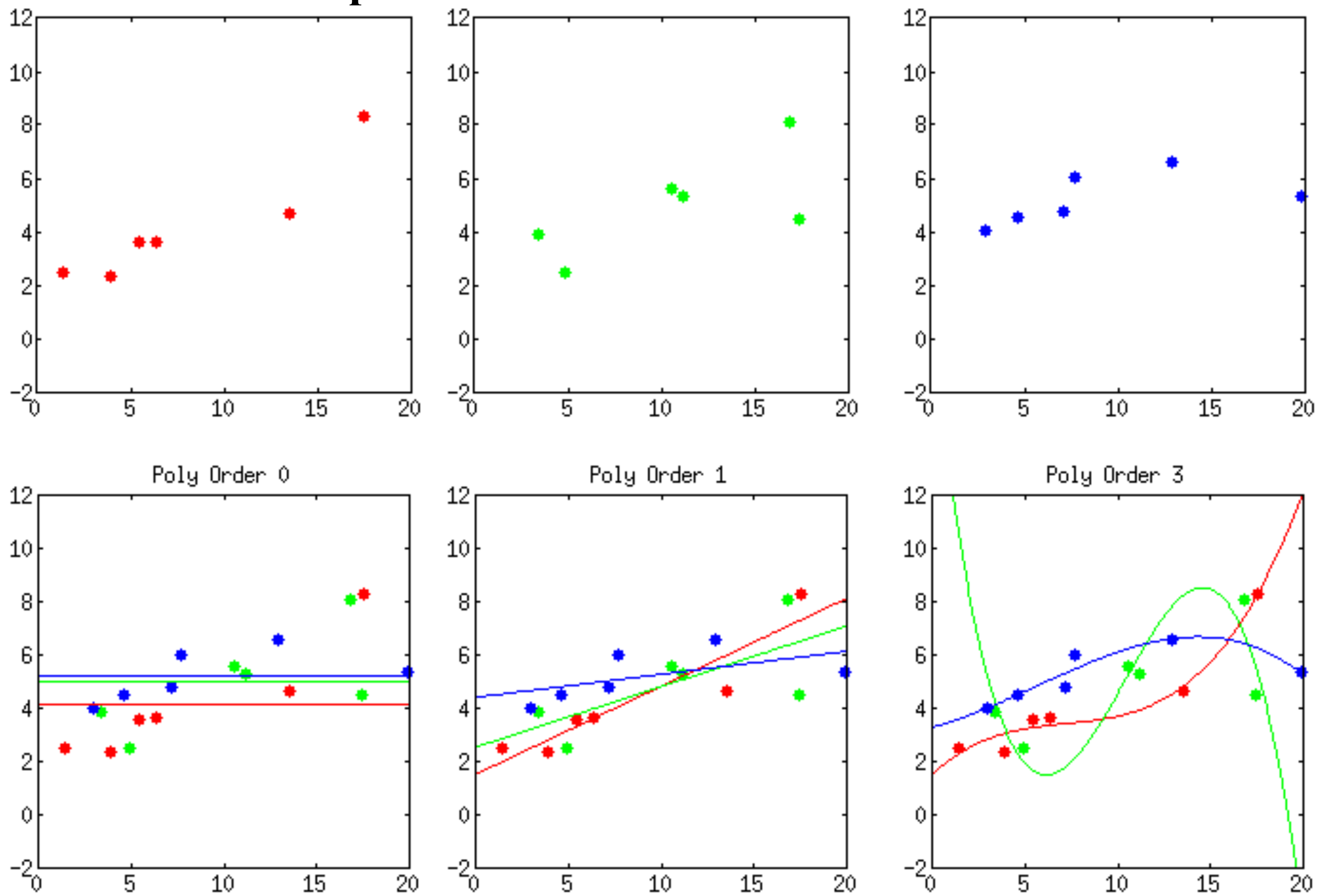


Bias & variance



Three different possible data sets:

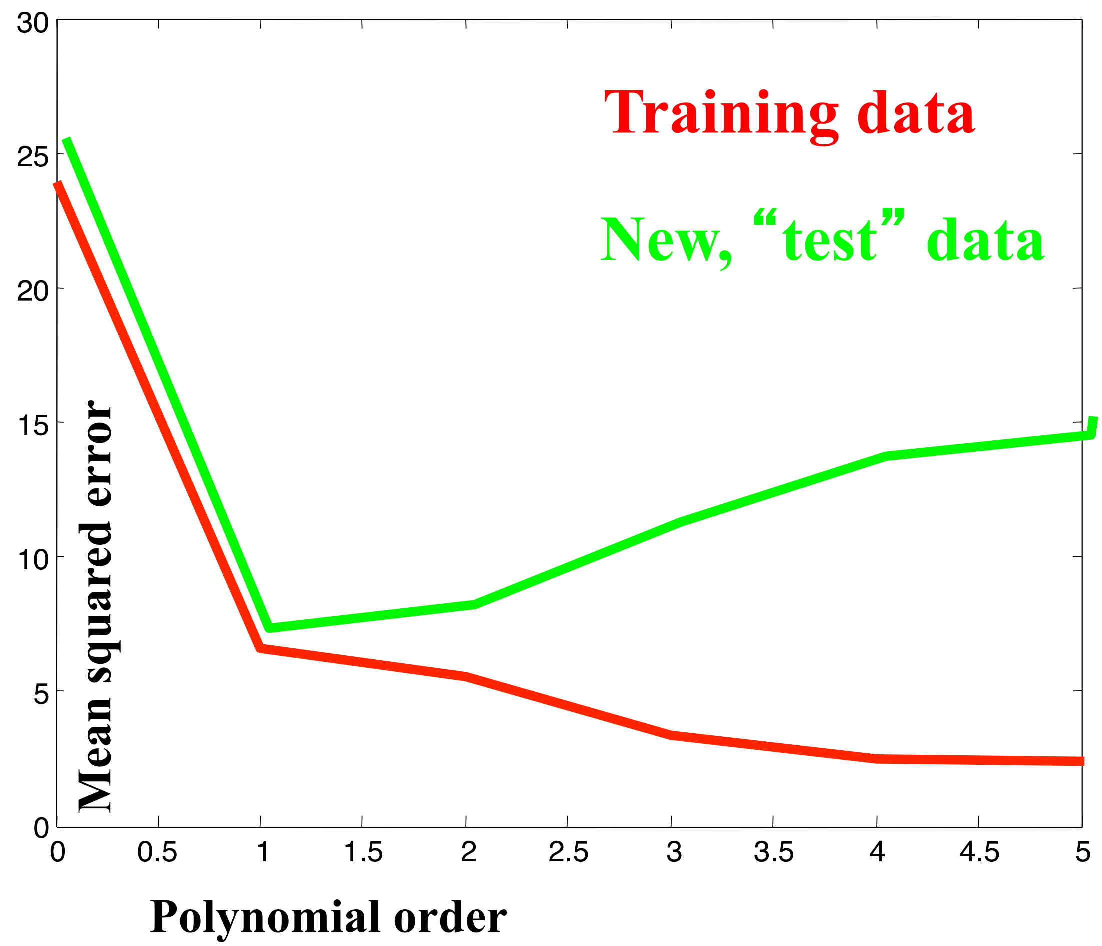
Each would give different predictors for any polynomial degree:



Training versus test error

- Plot SE as a function of model complexity
  - Polynomial order
- Decreases
  - More complex function fits training data better

- What about new data?
- 0^{th} to 1^{st} order
  - Error decreases
  - Underfitting
- Higher order
  - Error increases
  - Overfitting



Detecting overfitting

- Overfitting effect
  - Do better on training data than on future data
  - Need to choose the “right” complexity

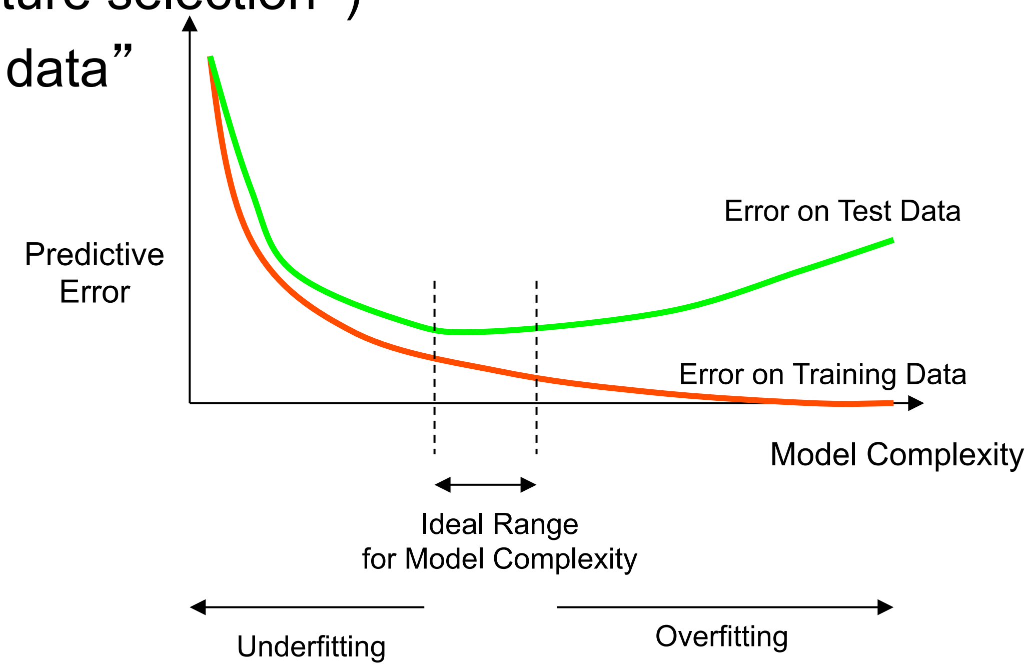
- One solution: “Hold-out” data
- Separate our data into two sets
  - Training
  - Test
- Learn only on training data
- Use test data to estimate generalization quality
  - Model selection

- All good competitions use this formulation
  - Often multiple splits: one by judges, then another by you

What to do about under/overfitting?

- Ways to increase complexity?
  - Add features, parameters
  - We'll see more…

- Ways to decrease complexity?
  - Remove features ("feature selection")
  - "Fail to fully memorize data"
    - Partial training
    - Regularization



Regularization

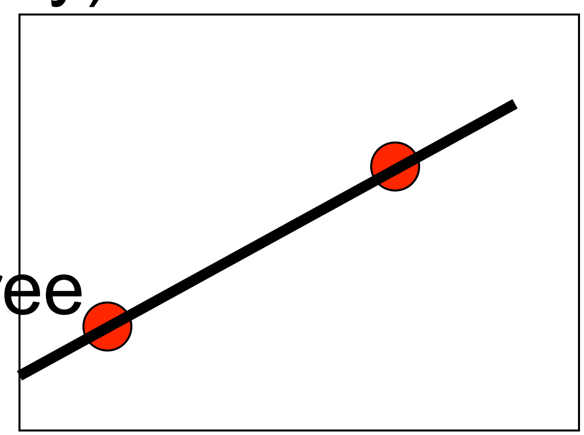
- Recall $$J(\underline{\theta}) = \frac{1}{2}(\underline{y} - \underline{\theta}\,\underline{X}^T) \cdot (\underline{y} - \underline{\theta}\,\underline{X}^T)^T$$

- Can add "preference" for certain parameters
  - Independent of the data

$$J(\underline{\theta}) = \frac{1}{2}(\underline{y} - \underline{\theta}\,\underline{X}^T) \cdot (\underline{y} - \underline{\theta}\,\underline{X}^T)^T + \alpha\theta\theta^T$$

- New solution (derive the same way)

$$\underline{\theta} \quad = \quad \underline{y}\,\underline{X}(\underline{X}^T\,\underline{X} + \alpha I)^{-1}$$



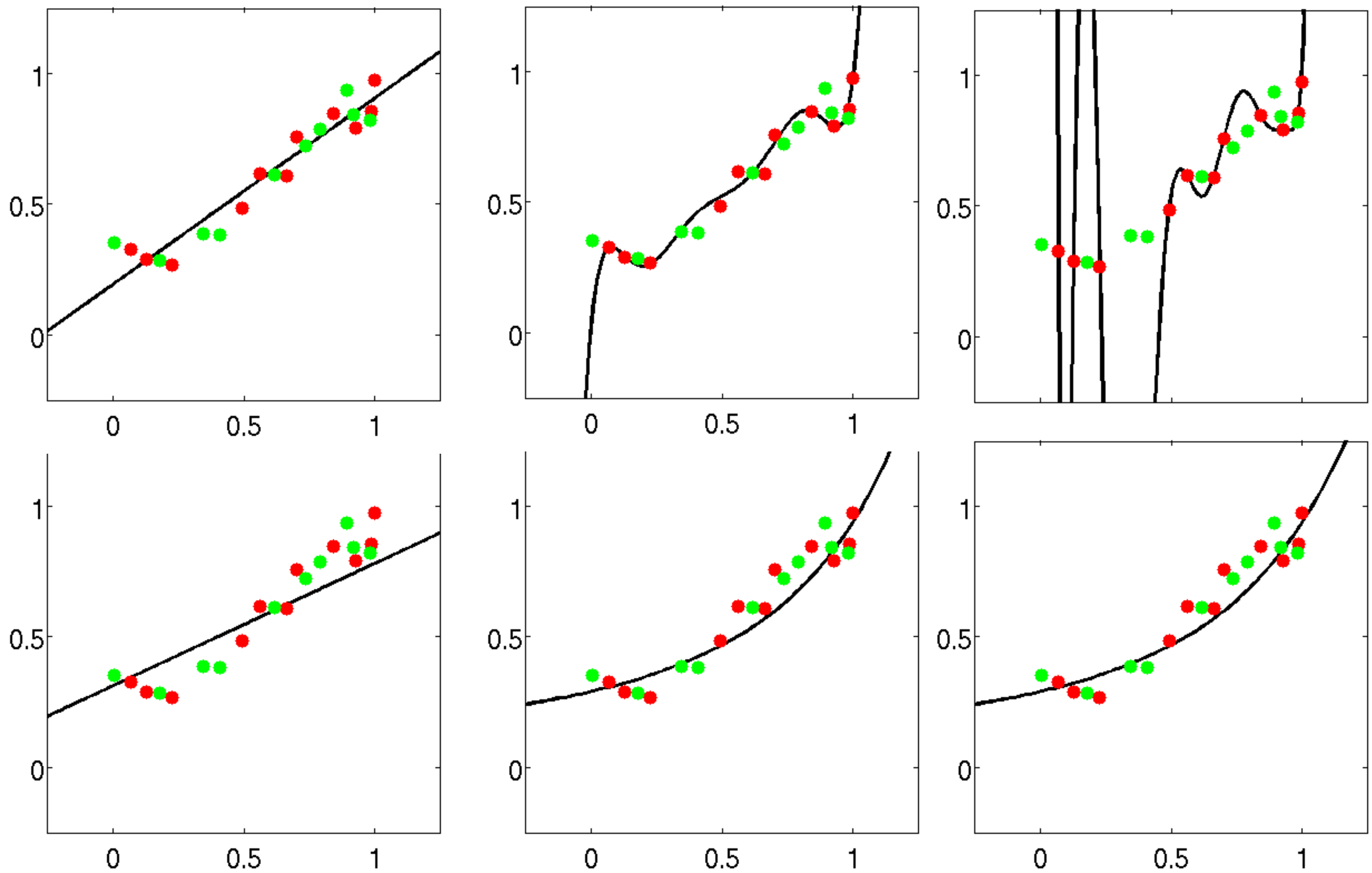
- Problem well-posed for any degree

Regularization

- Compare between unreg. & reg. results

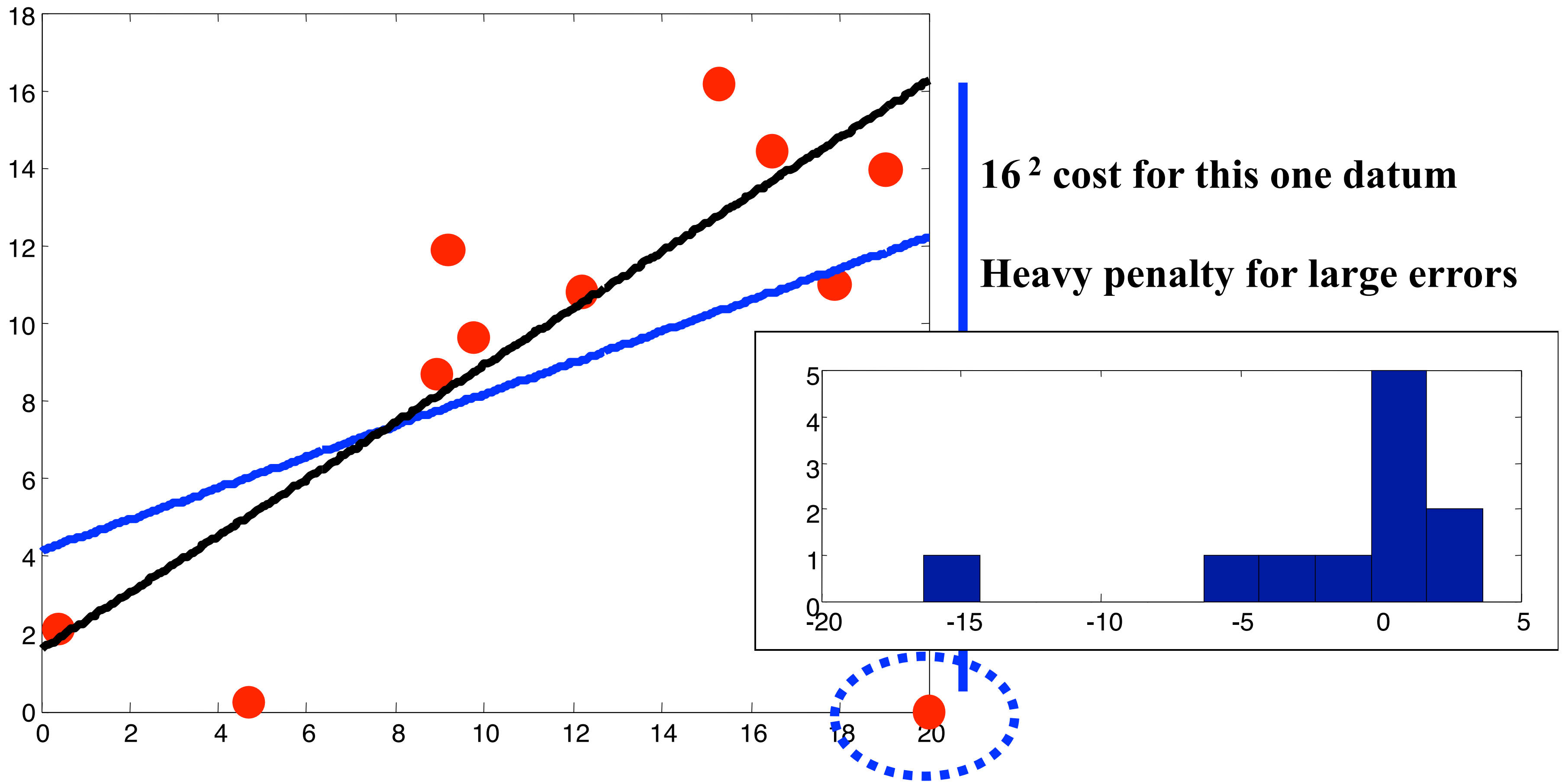
Alpha =0 (Unreg)

Alpha =1

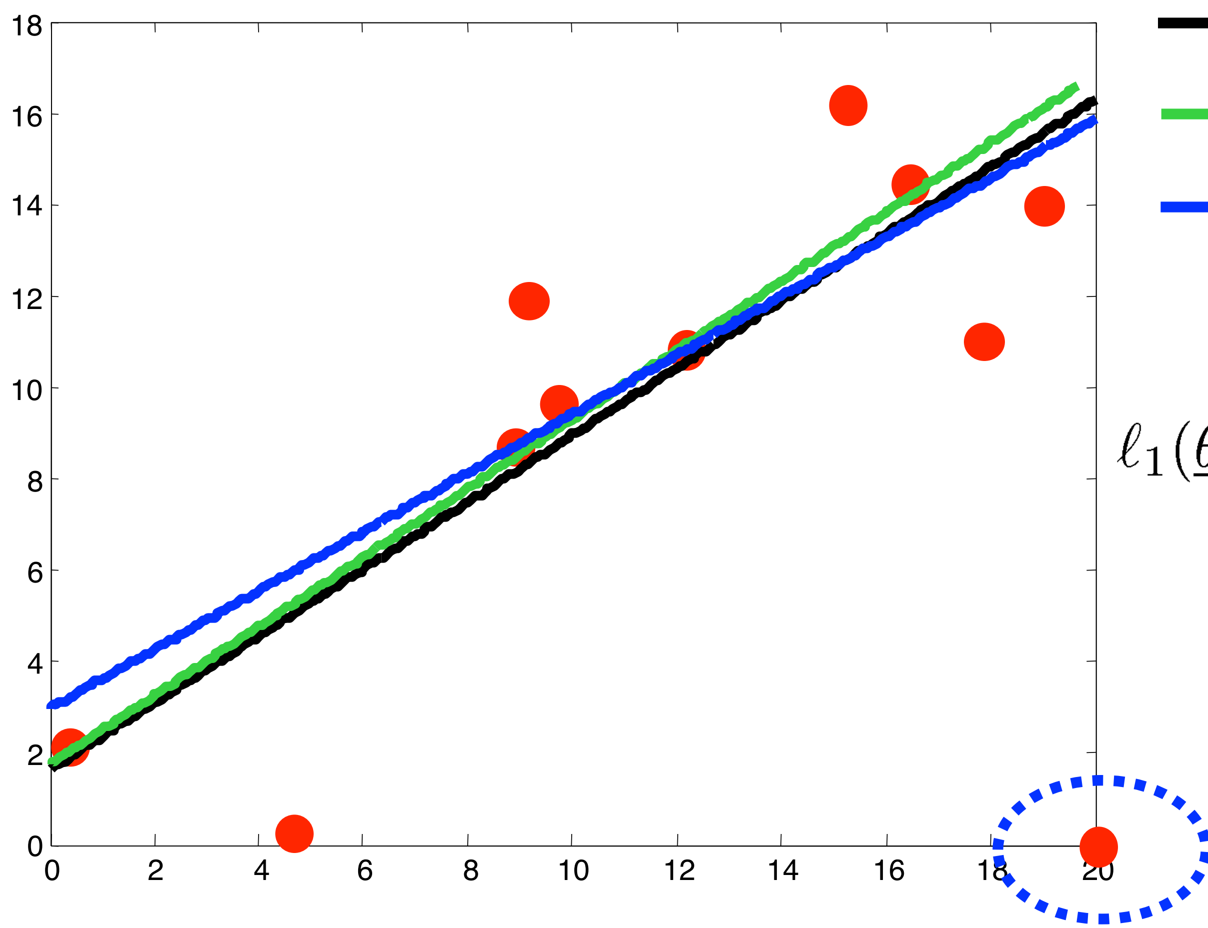


Effects of MSE choice

- Sensitivity to outliers



L1 error



$$\ell_1(\underline{\theta}) = \sum_j |y^{(j)} - \hat{y}(x^{(j)})|$$

$$= \sum_j |y - \underline{\theta} \cdot \underline{x}^T|$$

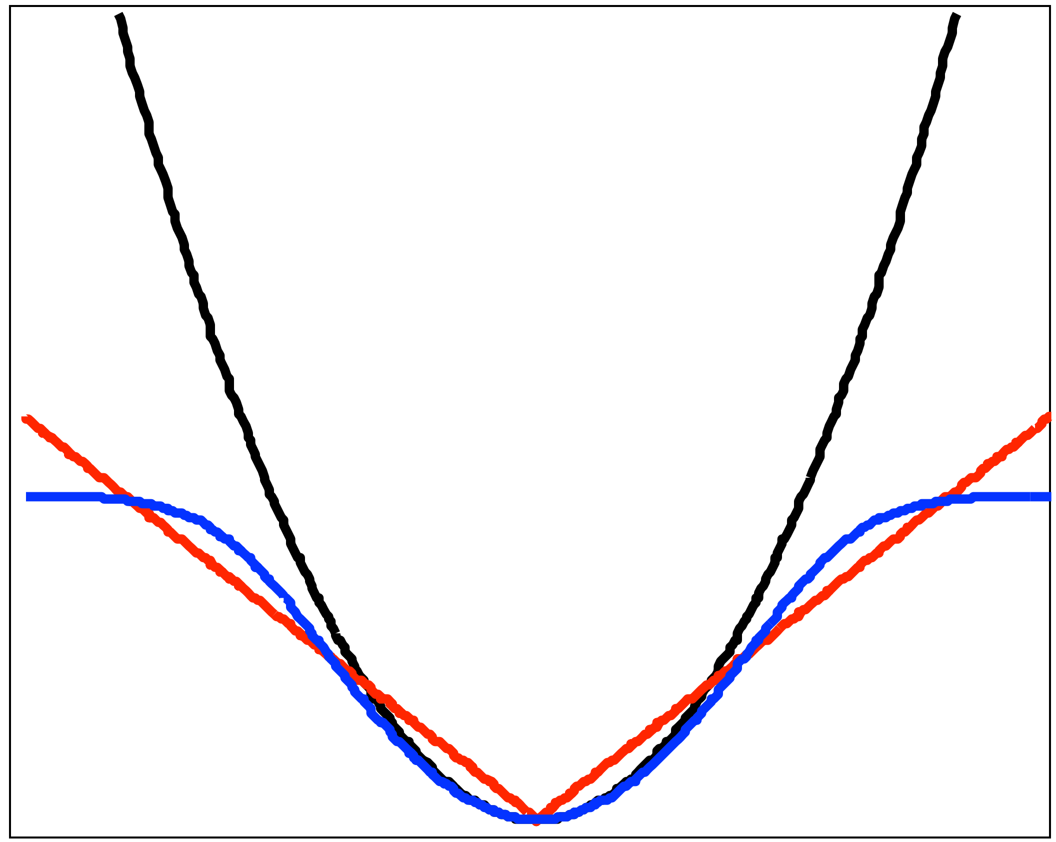
Robust cost functions

$$\ell_2 \ : \ (y - \hat{y})^2$$

$$\ell_1 \ : \ |y - \hat{y}|$$

Something else entirely…

$$c - \log(\exp(-(y - \hat{y})^2) + c)$$



$$\leftarrow (y - \hat{y}) \rightarrow$$

"Arbitrary" functions can't be solved in closed form…

- use gradient descent