Generalization

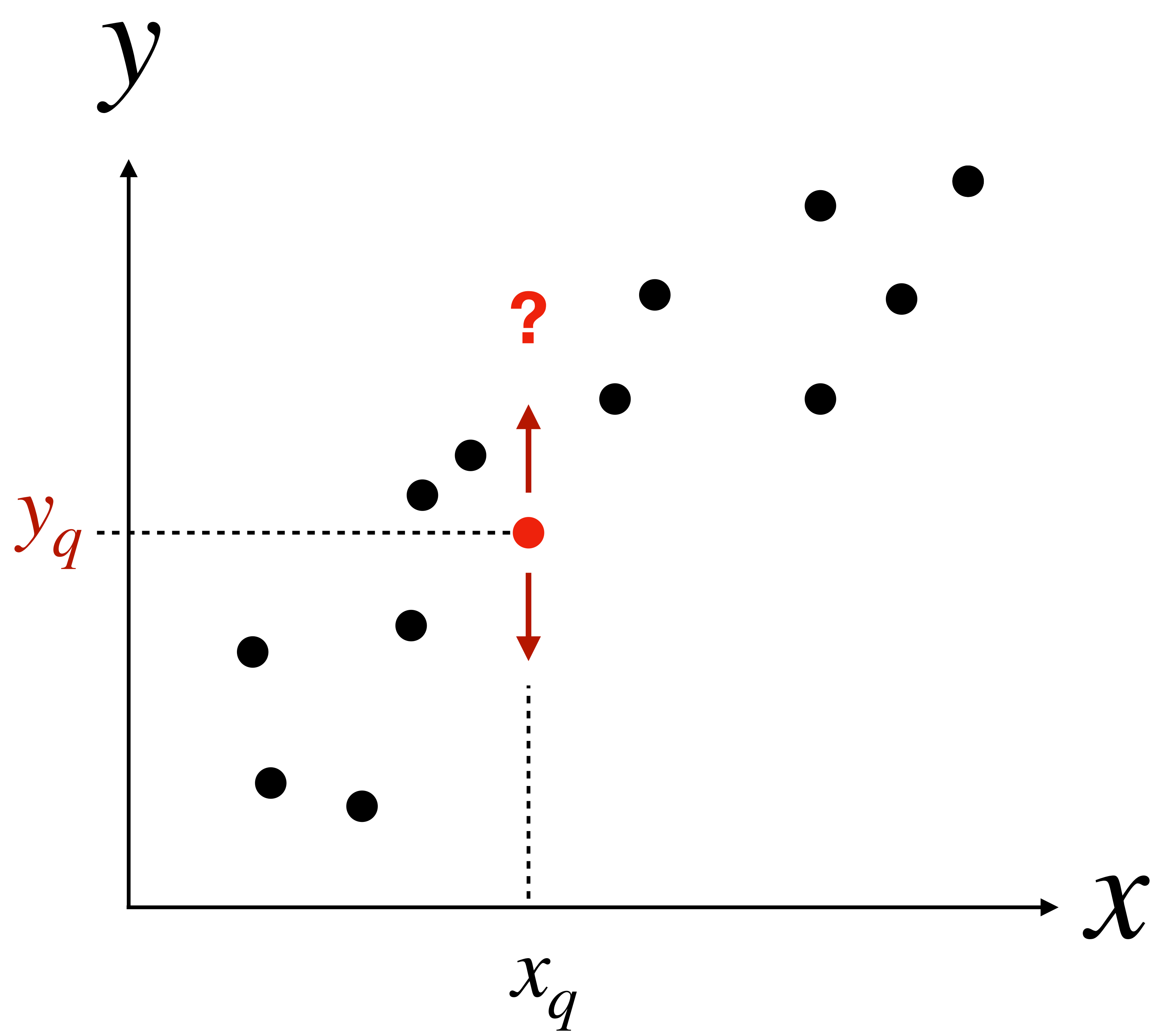
Prepared by: Joseph Bakarji


Compression
Measurements
Learning
World
Data
Modeling
Seen
Unseen
Inference
Projection
Reconstruction

Given new input, what's the output?

Input	Output
x	y
$x^{(1)}$	$y^{(1)}$
$x^{(2)}$	$y^{(2)}$
$x^{(3)}$	$y^{(3)}$
$x^{(4)}$	$y^{(4)}$
$\vdots$	$\vdots$

Query x_q ?? Prediction



Linear Regression

1. Assume a linear hypothesis

$$h_\theta(\mathbf{x}) = \theta^\top \mathbf{x} = \sum_{i=0}^{d} \theta_i \, x_i$$

2. Cost function

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{d} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2$$

3. Minimize: Gradient Descent

$$\theta_i := \theta_i - \alpha \frac{\partial J(\theta)}{\partial \theta_i}$$

SGD

for t = 1…T:

  for i = 1…n:

$$\theta := \theta - \alpha \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x^{(i)}$$

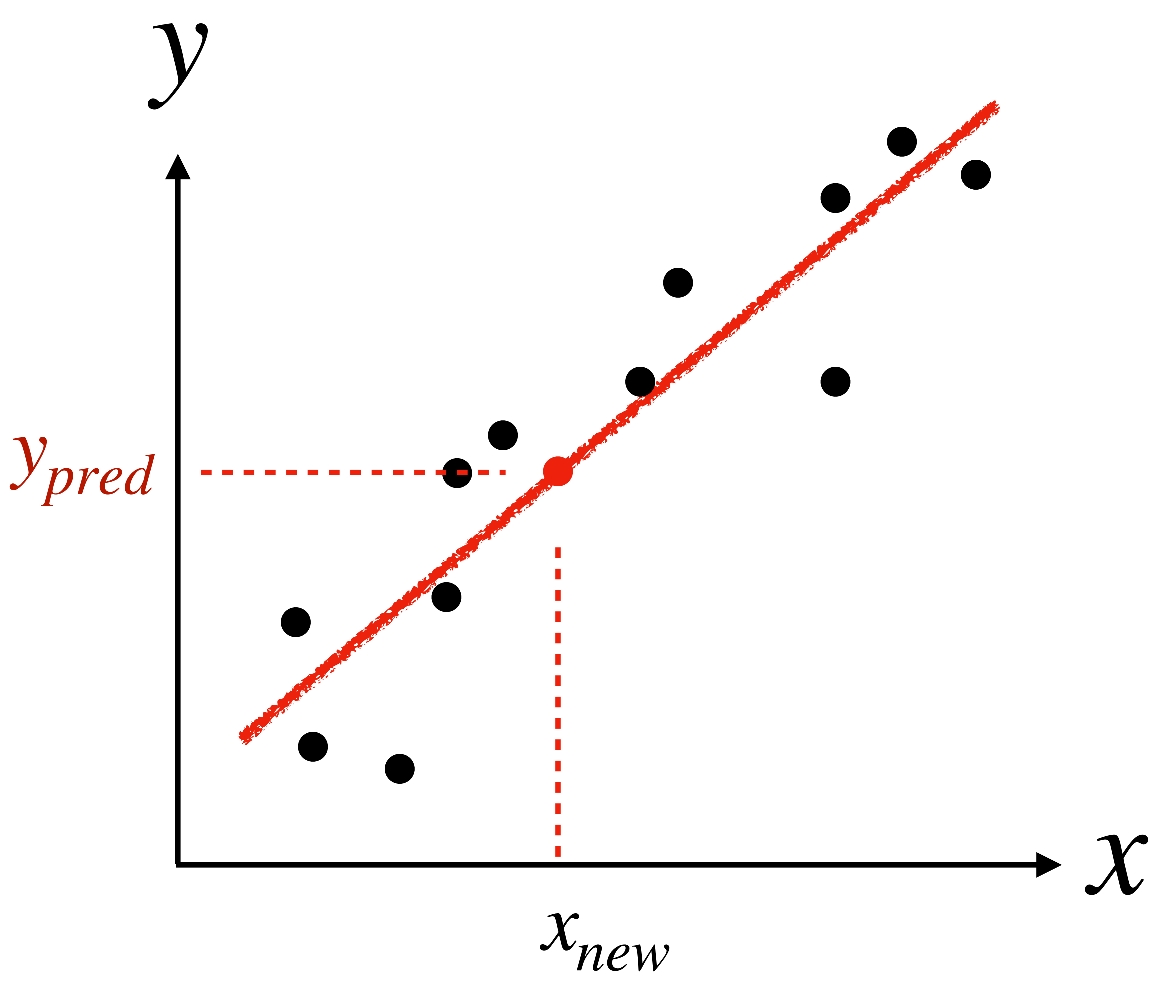
$\hat{\theta}$

4. Optimal predictor

$$y = h_{\hat{\theta}}(x)$$

5. Predict unseen data

$$y_{pred} = h_{\hat{\theta}}(x_{new})$$

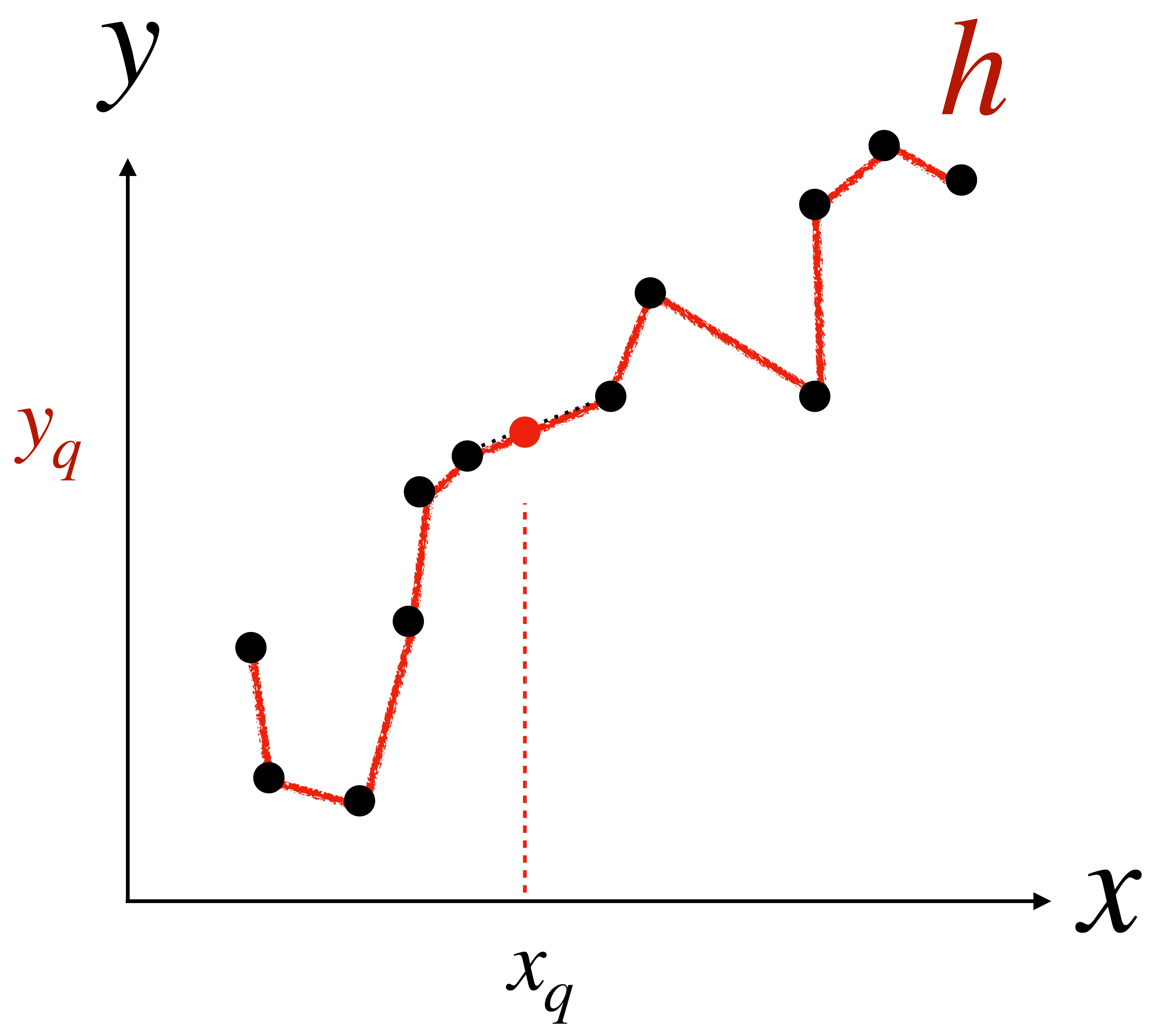


Given new input, what's the output?

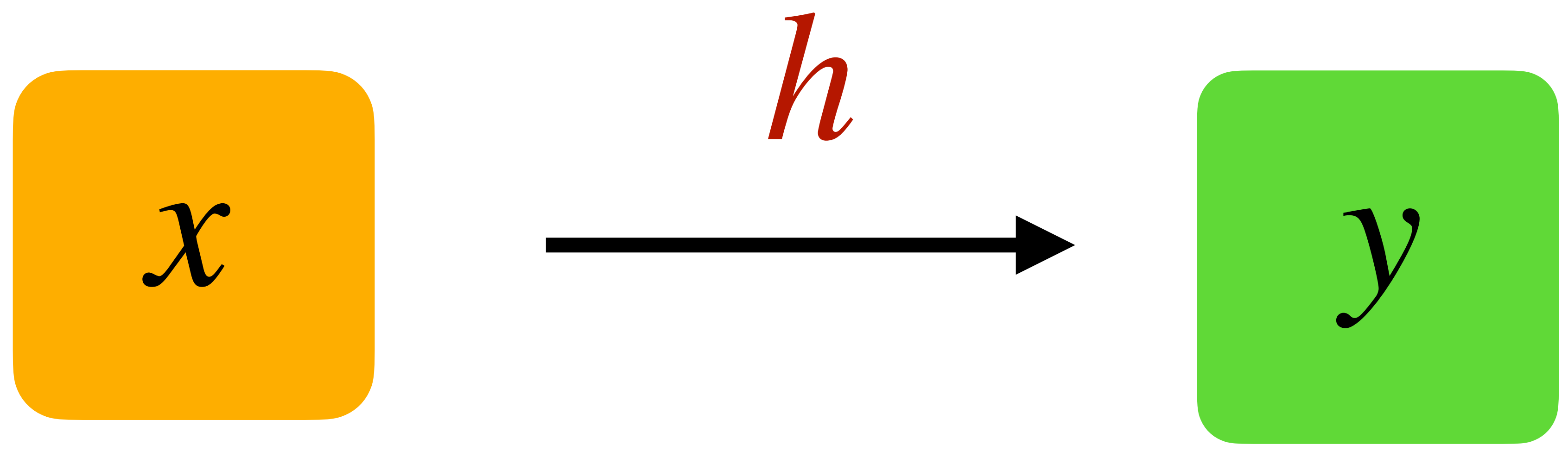
$$x \xrightarrow{h} y$$

Given the data,
find a **function** h, also called a **hypothesis,**
that predicts an output, given an input

Linear Interpolation

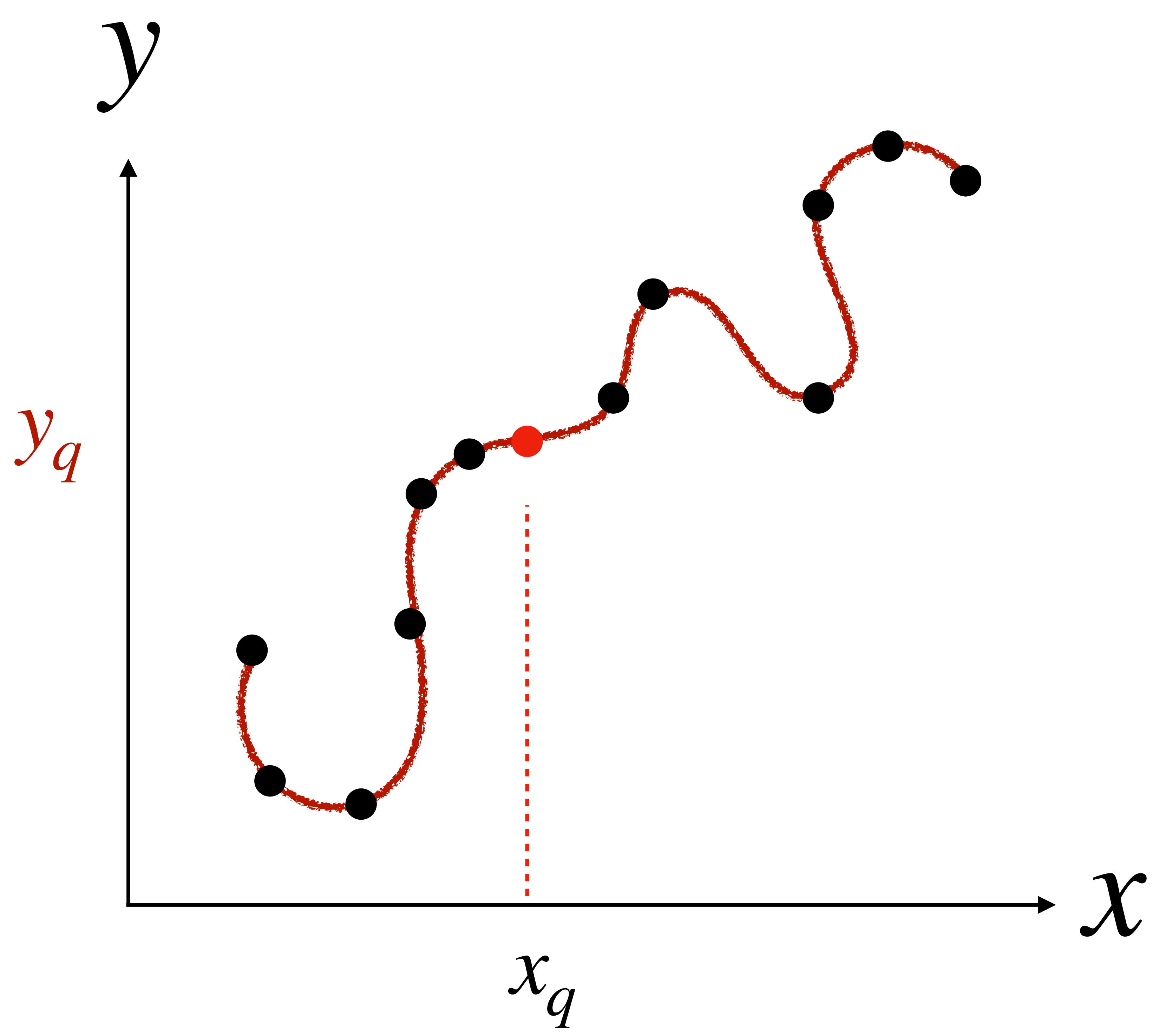


Given new input, what’s the output?

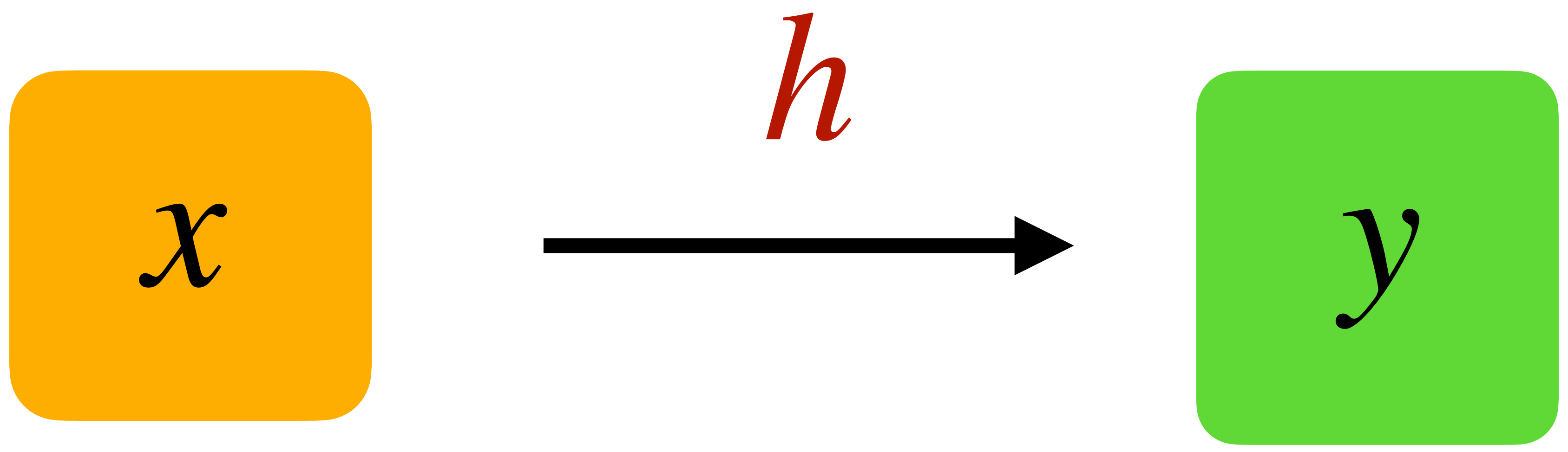


Given the data,
find a **function** h, also called a **hypothesis,**
that predicts an output, given an input

Polynomial Interpolation

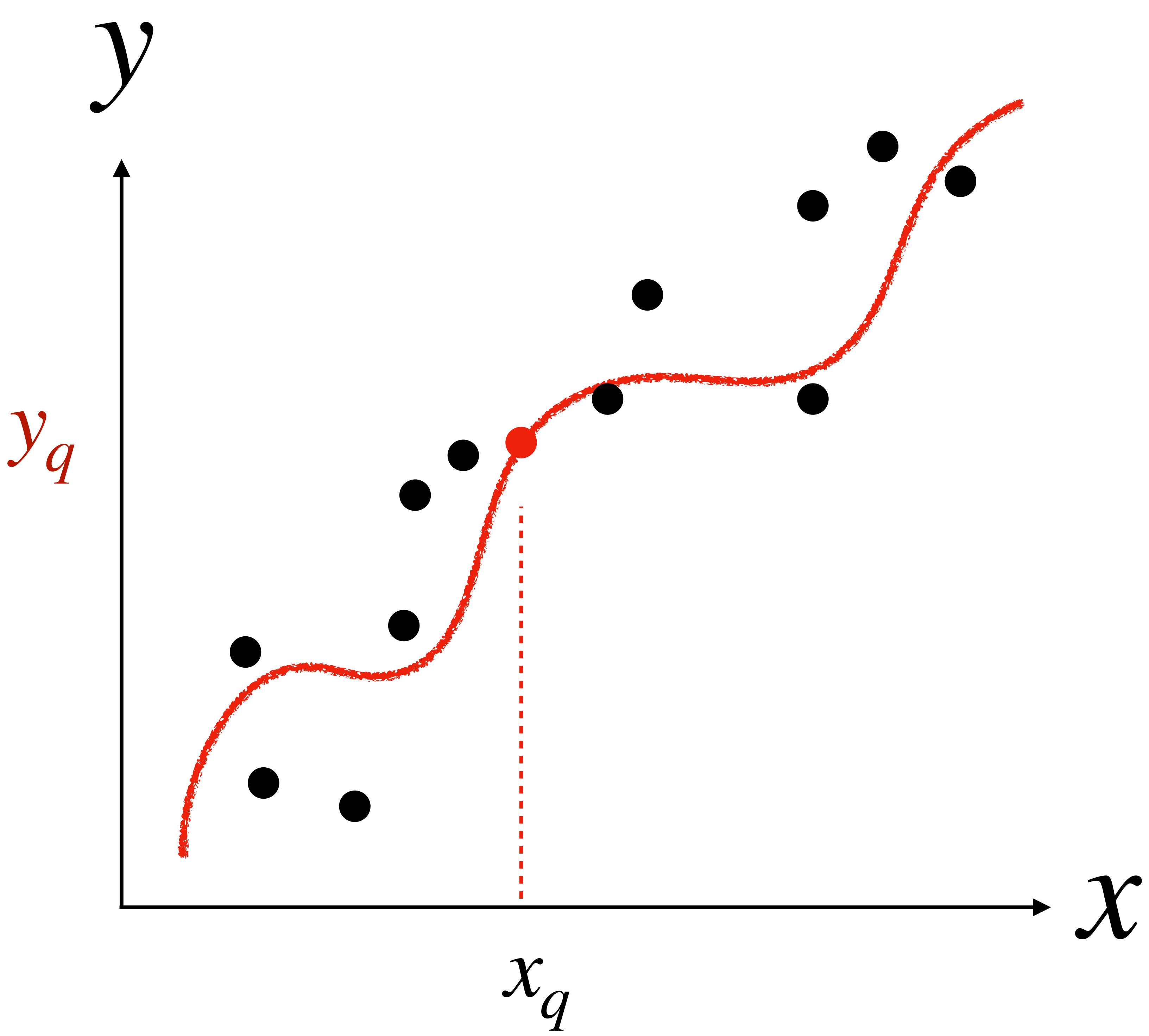


Given new input, what’s the output?



Given the data,
find a **function** h, also called a **hypothesis,**
that predicts an output, given an input

Some other function?

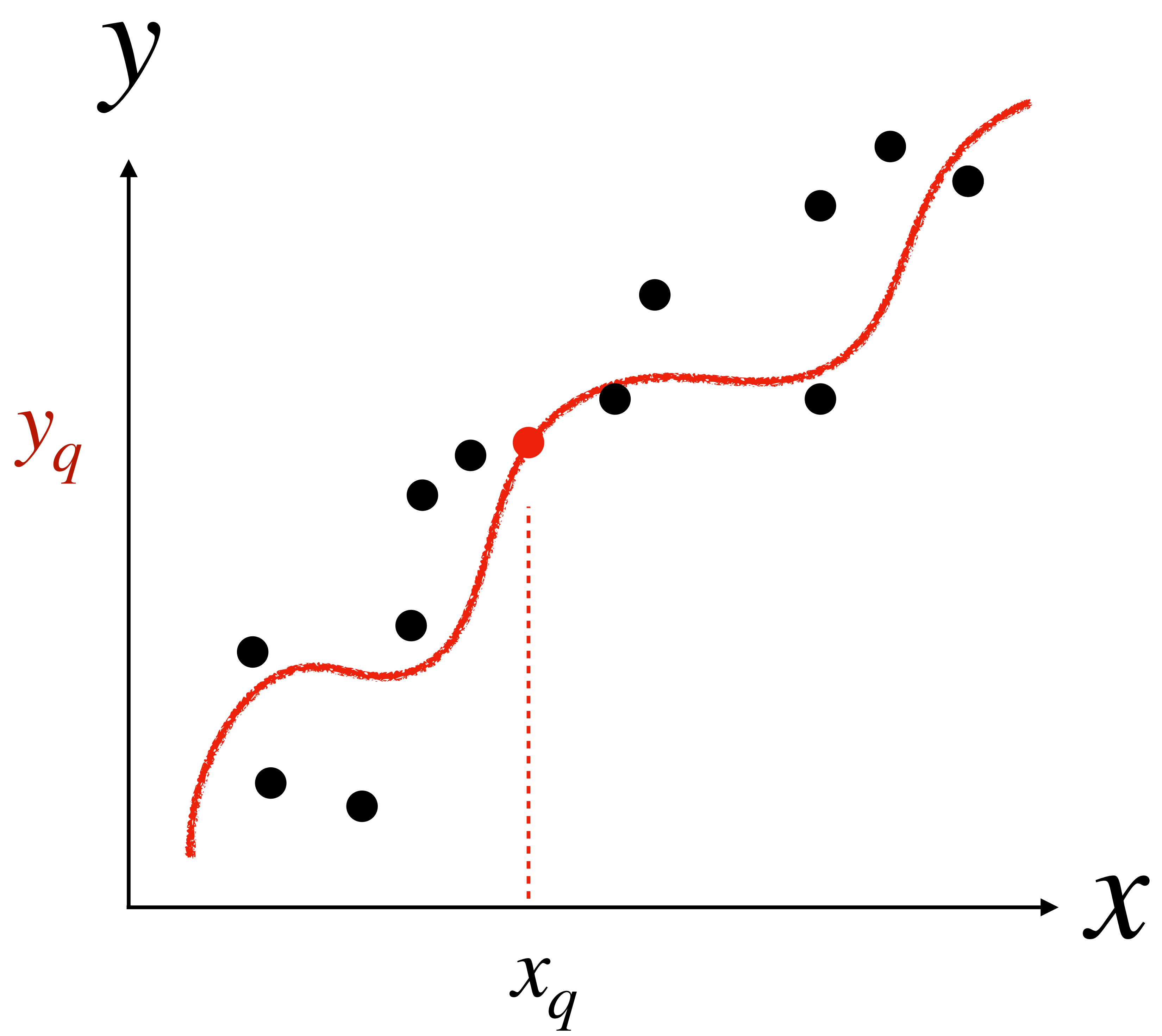


How do you **choose the hypothesis**?

$$x \xrightarrow{h} y$$

Given the data,
find a **function** h, also called a **hypothesis,**
that predicts an output, given an input

Some other function?



What happens if we have more inputs?

Assume a linear hypothesis

$$\mathbf{x} \xrightarrow{h} y$$

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

$$h_\theta(\mathbf{x}) = \underbrace{[\theta_0, \theta_1, \theta_2, \theta_3, \ldots]}_{\theta} \cdot \underbrace{[1, x_1, x_2, x_3, \ldots]}_{\mathbf{x}}$$

weights

inputs

$$h_\theta(\mathbf{x}) = \theta \cdot \mathbf{x} = \theta^\top \mathbf{x}$$

Inputs		Output
x_1	x_2	y
$x_1^{(1)}$	$x_2^{(1)}$	$y^{(1)}$
$x_1^{(2)}$	$x_2^{(2)}$	$y^{(2)}$
$x_1^{(3)}$	$x_2^{(3)}$	$y^{(3)}$
$x_1^{(4)}$	$x_2^{(4)}$	$y^{(4)}$
$\vdots$	$\vdots$	$\vdots$

Feature Engineering

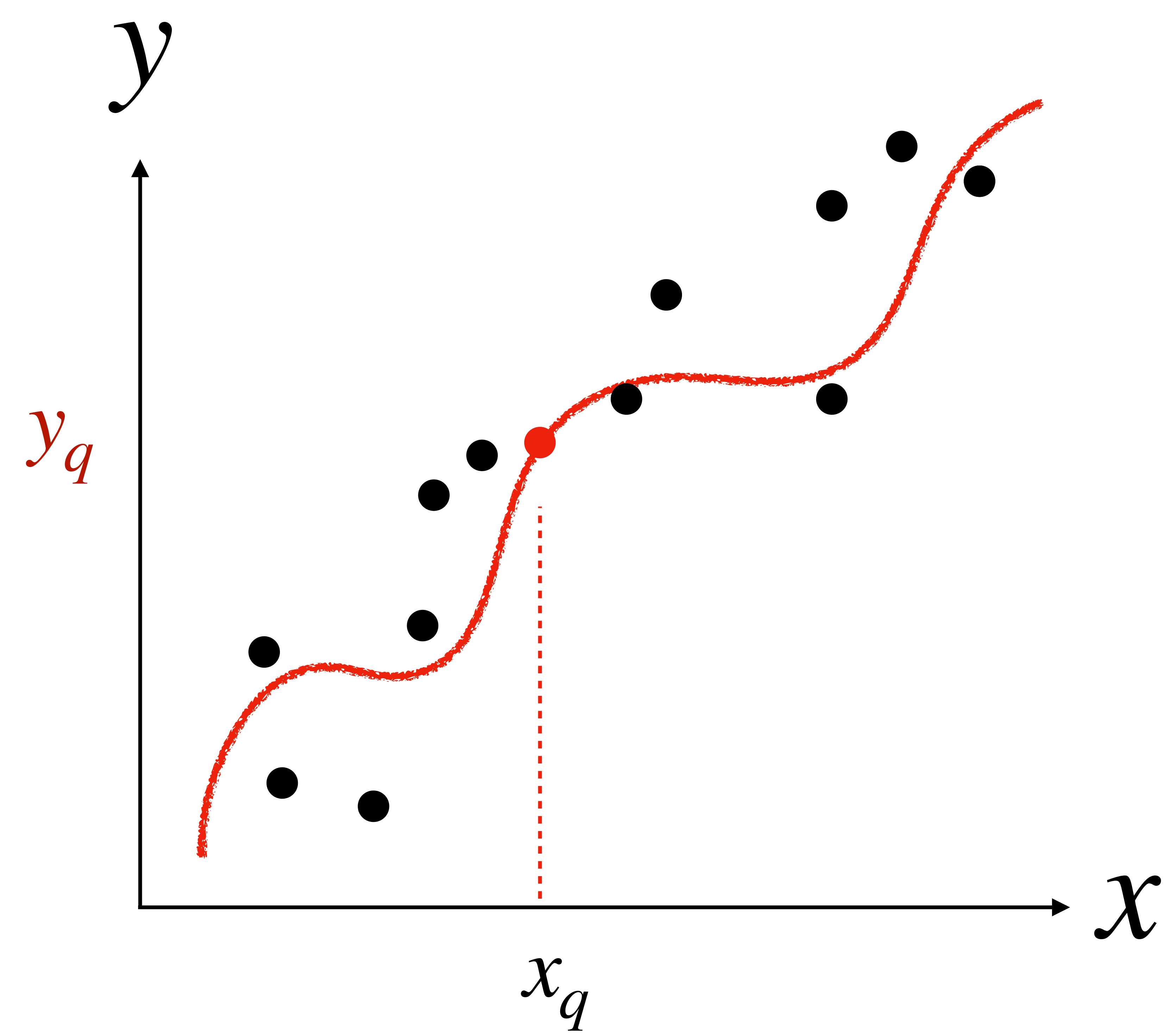
h **is does not have to be linear in** x

Input	Output
x	y
$x^{(1)}$	$y^{(1)}$
$x^{(2)}$	$y^{(2)}$
$x^{(3)}$	$y^{(3)}$
$x^{(4)}$	$y^{(4)}$

Example: construct a polynomial model

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \ldots$$

Some other function?



$$h_\theta(x) = \underbrace{[\theta_0, \theta_1, \theta_2, \theta_3, \ldots]}_{\theta} \cdot \underbrace{[1, x, x^2, x^3, \ldots]}_{\phi(x)}$$

Feature map

$$h_\theta(x) = \theta^\top \phi(x)$$

Feature Engineering

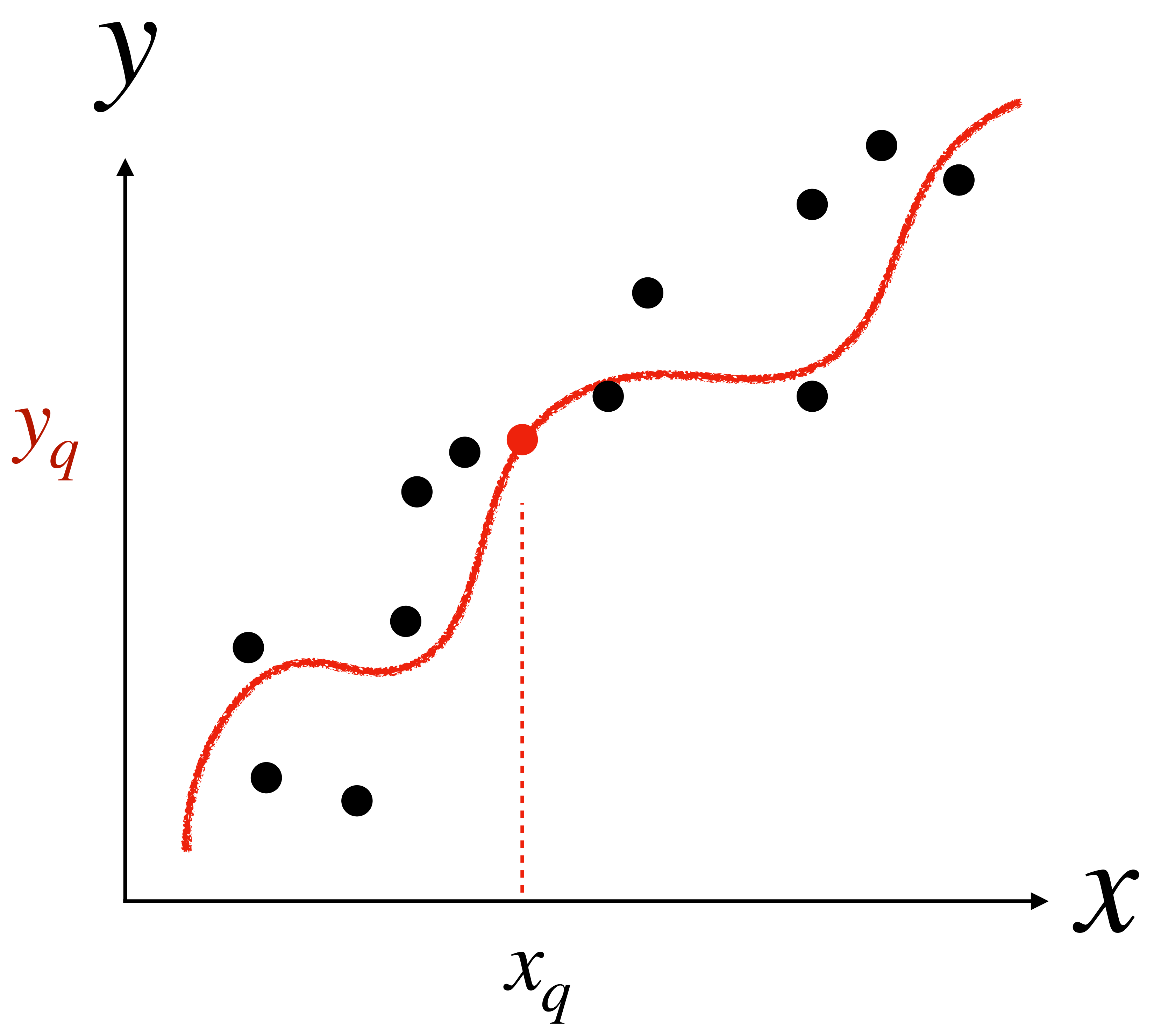
h **is does not have to be linear in** x

Input x	Output y
$x^{(1)}$	$y^{(1)}$
$x^{(2)}$	$y^{(2)}$
$x^{(3)}$	$y^{(3)}$
$x^{(4)}$	$y^{(4)}$

Example: construct a polynomial model

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \ldots$$

Some other function?



$$h_\theta(x) = \underbrace{[\theta_0, \theta_1, \theta_2, \theta_3, \ldots]}_{\theta} \cdot \underbrace{[1, x, x^2, x^3, \ldots]}_{\phi(x)}$$

Feature map

$$h_\theta(x) = \theta^\top \phi(x) = \theta_0 \phi_0(x) + \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \ldots$$

Feature Engineering

A feature map can also **drop features**

Example: construct a polynomial model

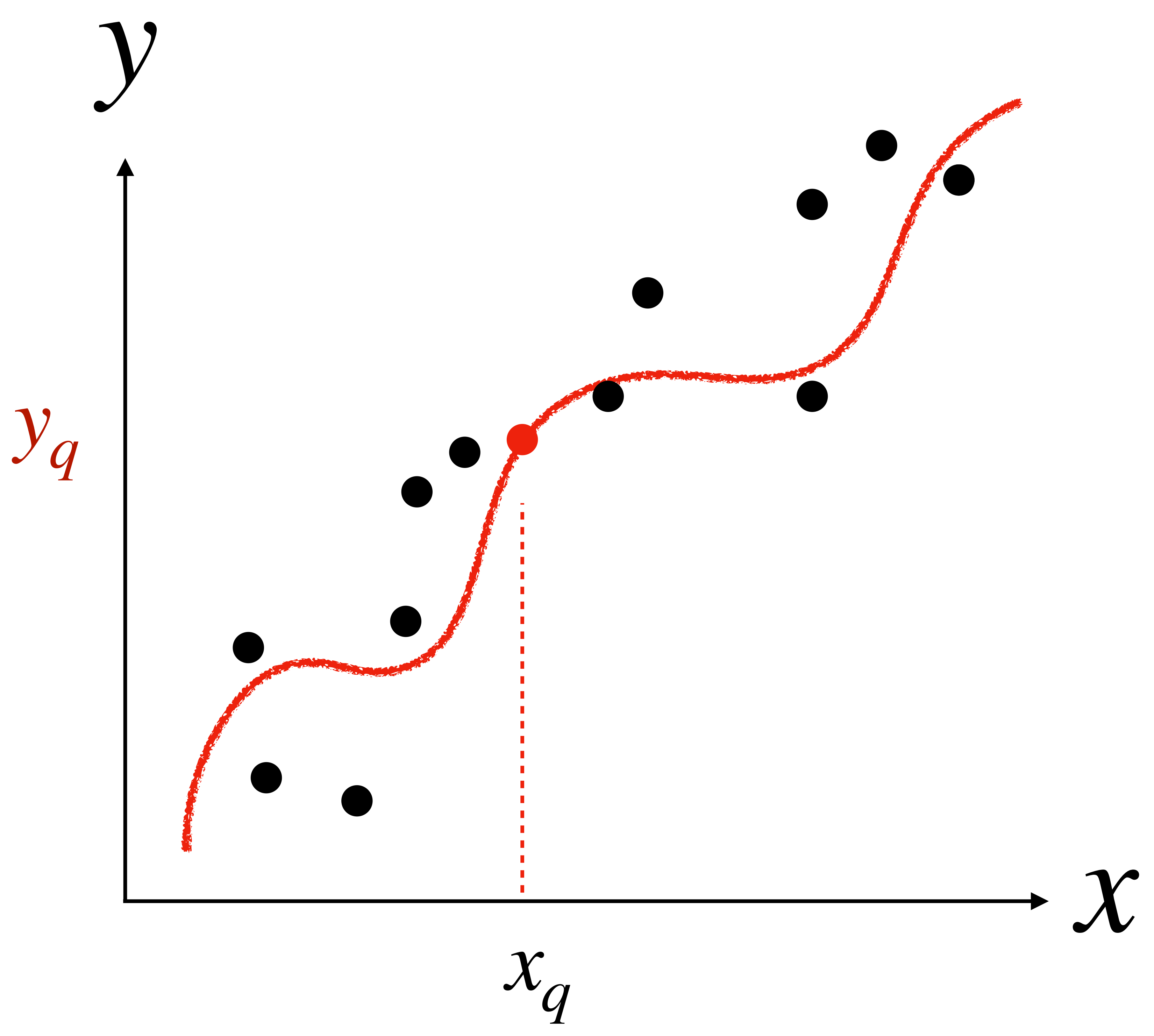
$$h_\theta(x) = \theta_1 x + \theta_3 x^3$$

$$h_\theta(x) = [\theta_1, \theta_3] \cdot \underbrace{[x,\ x^3]}_{\phi(x)}$$

Feature map

$$h_\theta(x) = \theta^\top \phi(x)$$

Some other function?

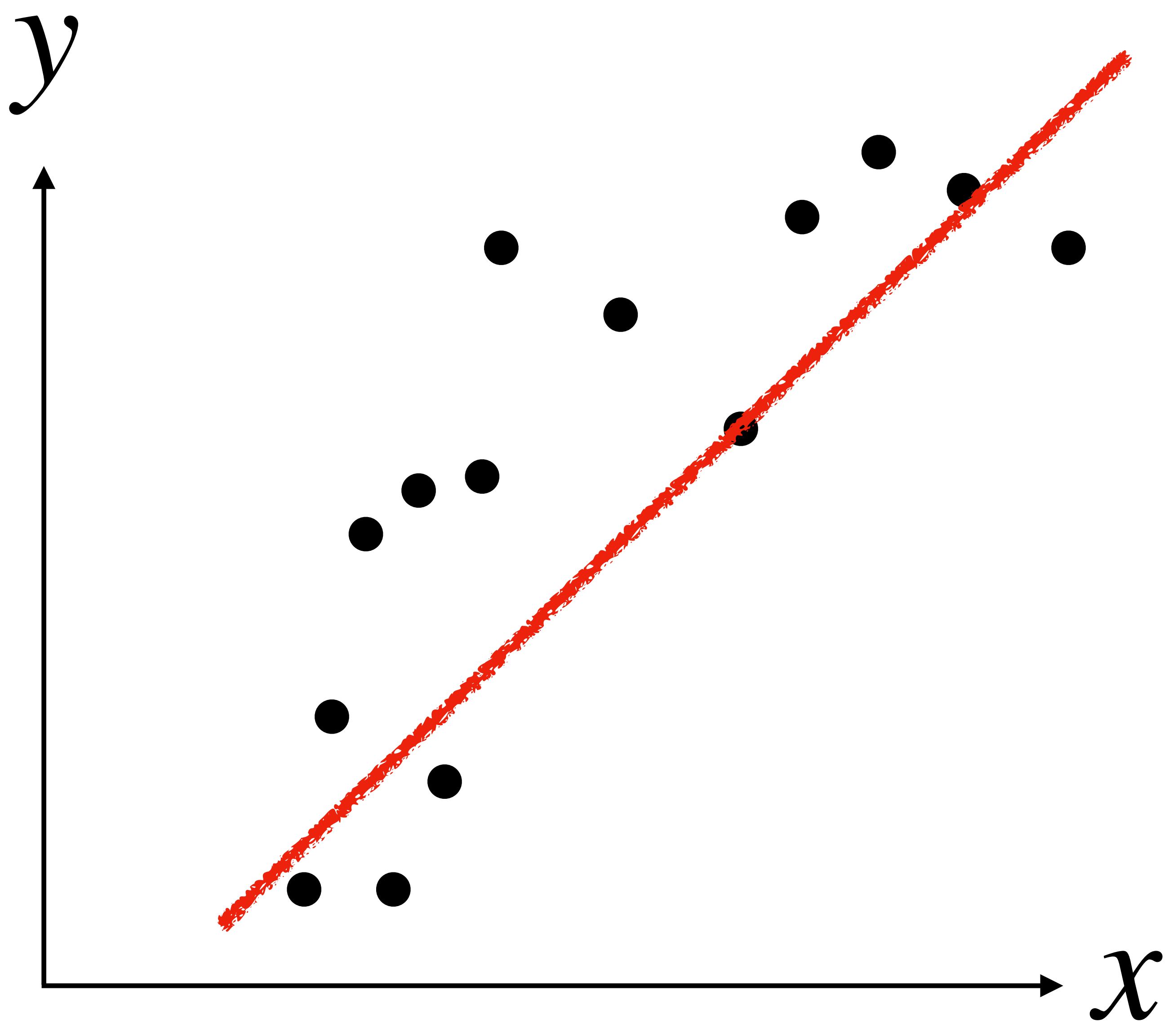


How to choose $\phi(x)$?

How to optimize over $\phi(x)$

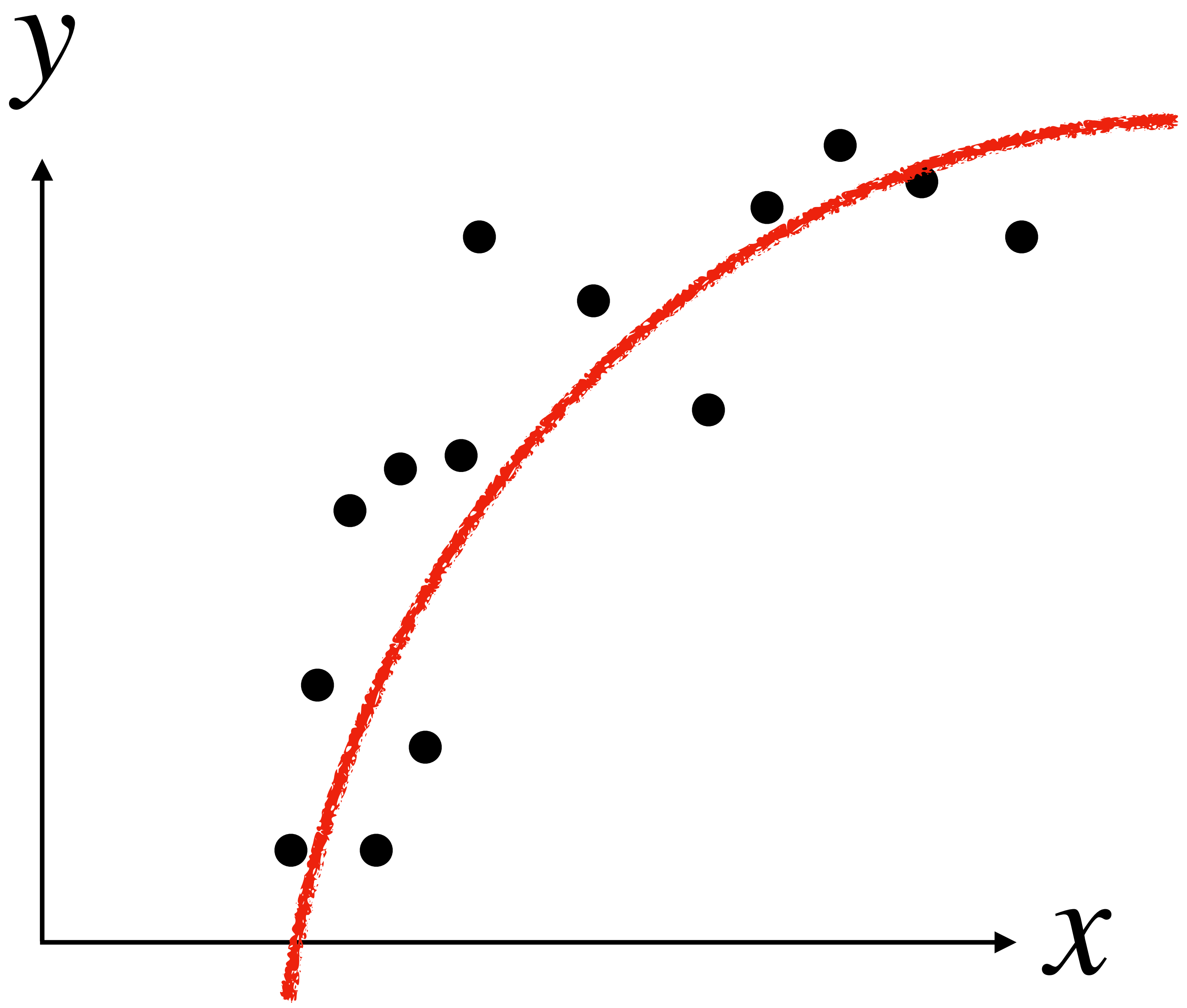
Underfitting

High Bias



$\phi(x) = [1, x]$

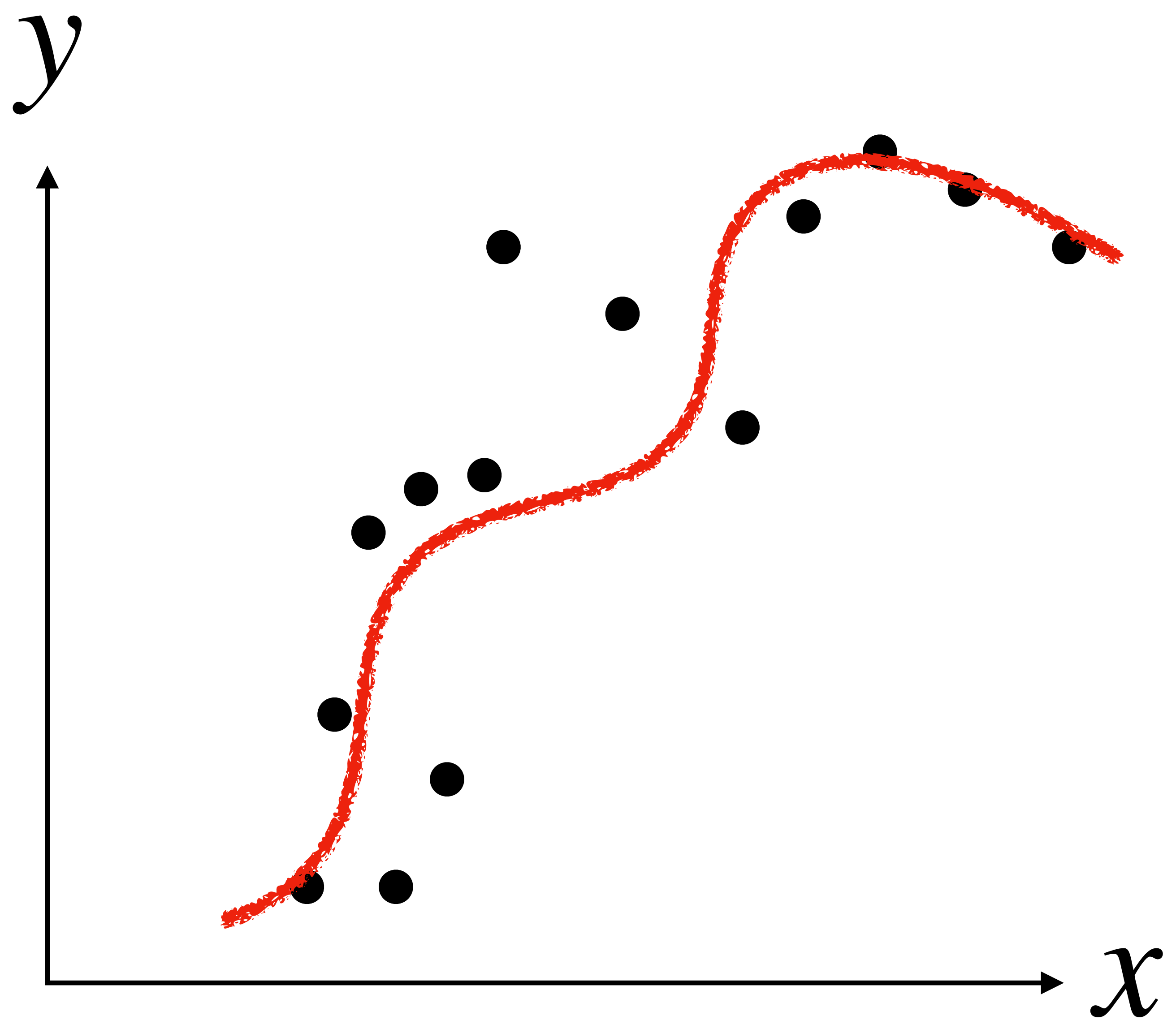
Just right



$\phi(x) = [1, x, x^2]$

Overfitting

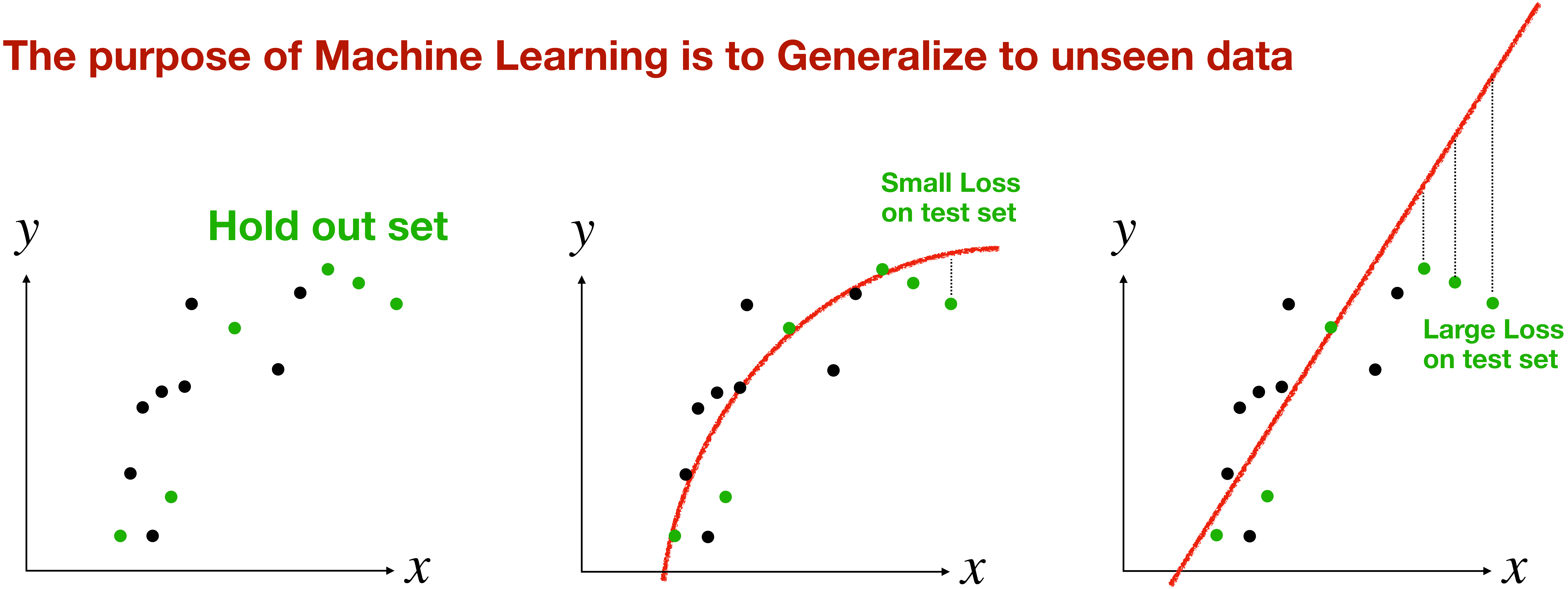
High Variance



$\phi(x) = [1, x, x^2, x^3, \ldots]$

How can we tell if $\phi(\cdot)$ is good?

The purpose of Machine Learning is to Generalize to unseen data



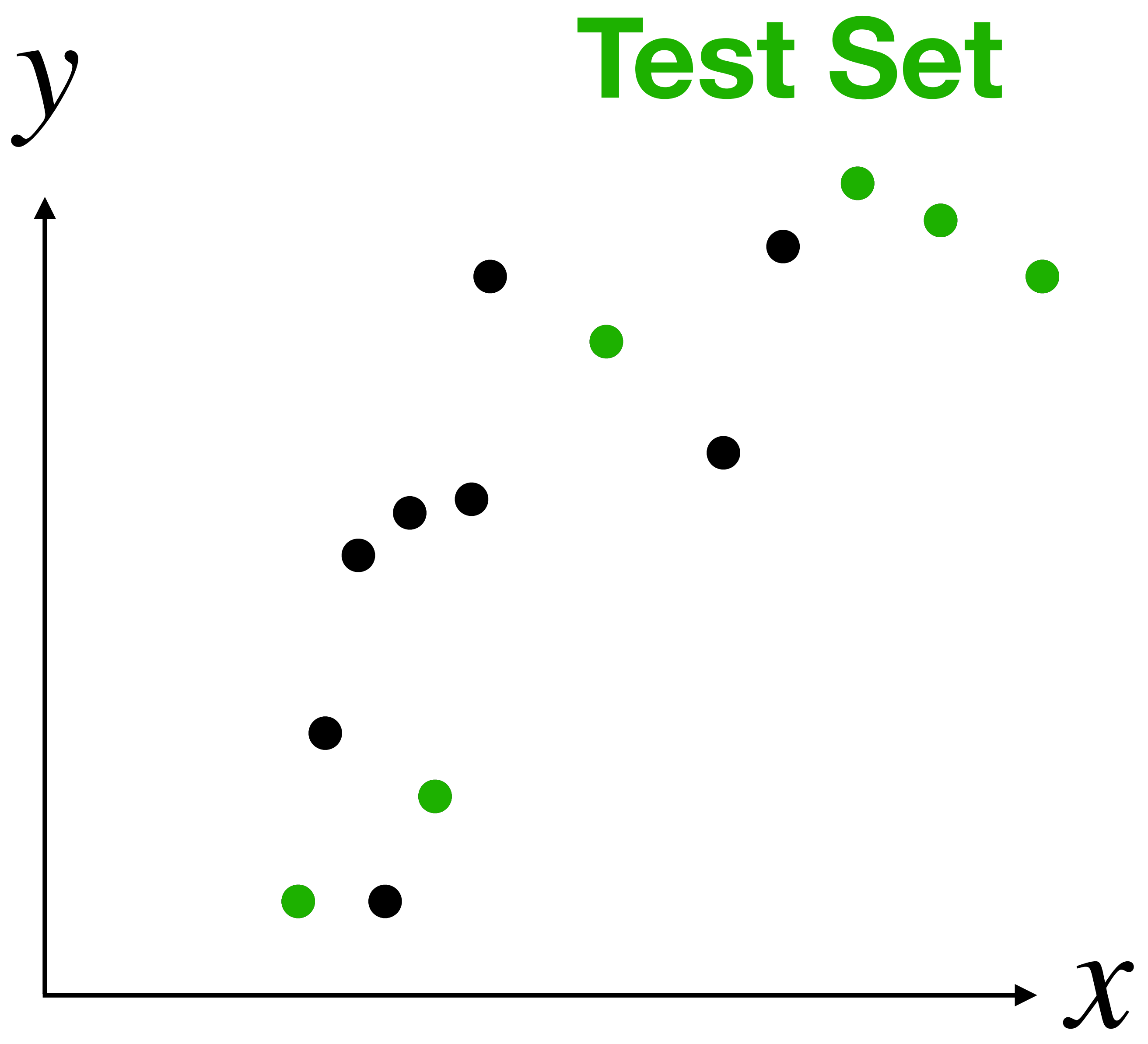
Create a test set to evaluate model

$\phi(x) = [1, x, x^2]$

$\phi(x) = [1, x]$

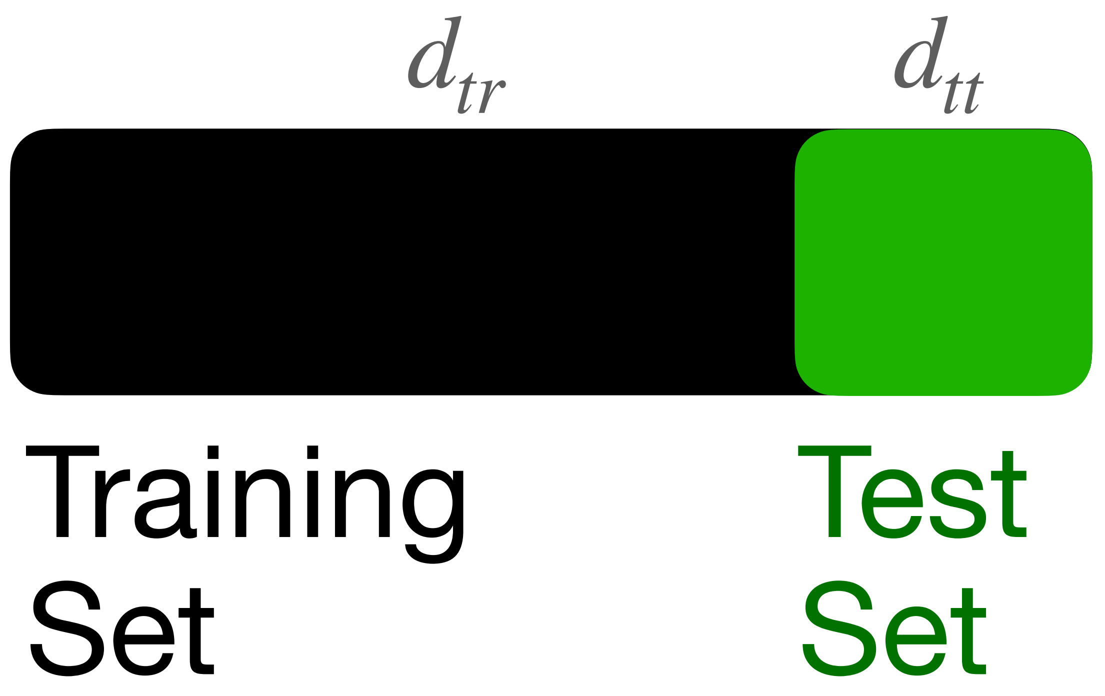
How do we tell that $\phi(\,\cdot\,)$ is good?

Define objective functions for each subset



Create a Test set to evaluate model

Split data:

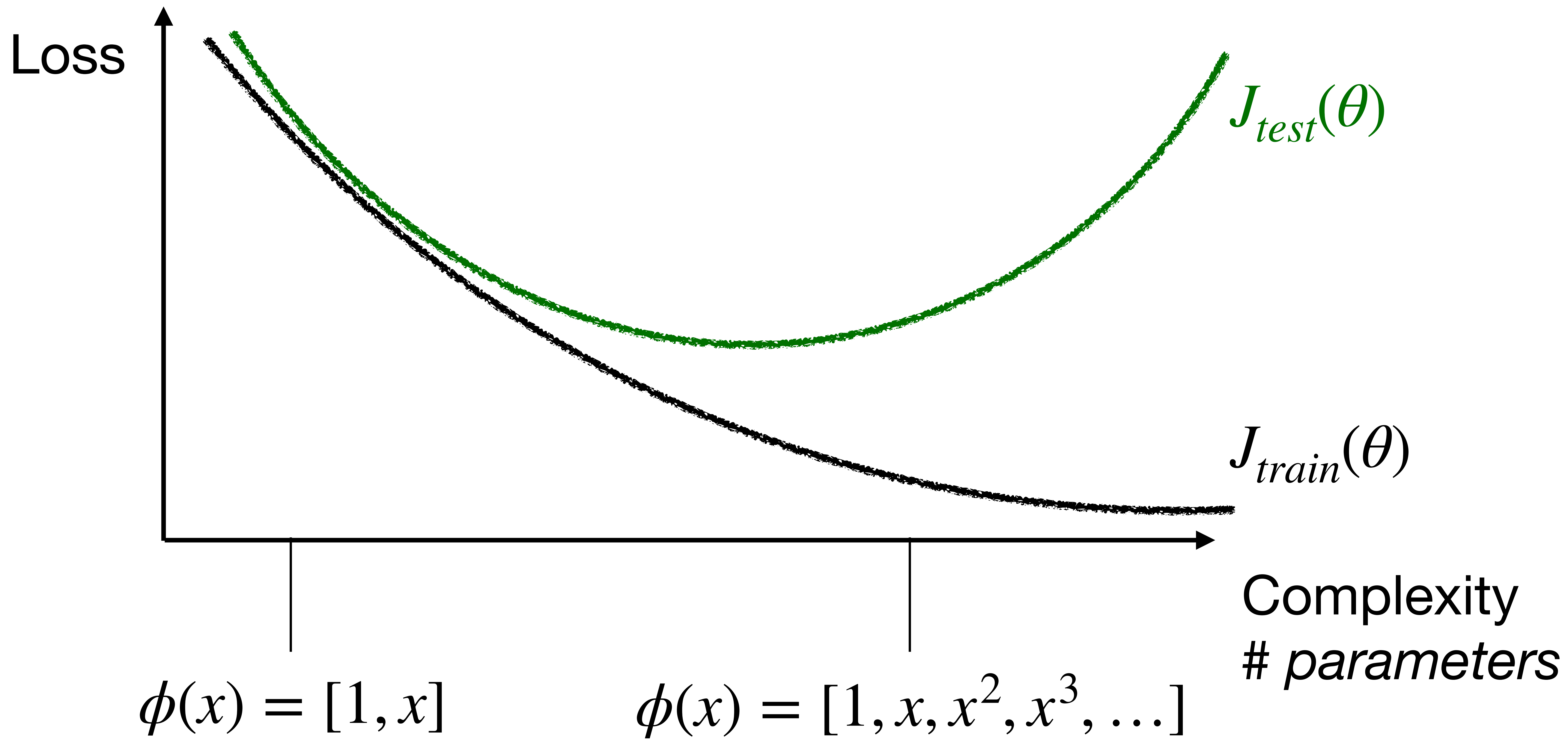


$$J_{train}(\theta) = \frac{1}{2d_{tr}} \sum_{i=1}^{d_{tr}} \left(\theta^\top \phi(x^{(i)}) - y^{(i)}\right)^2$$

$$J_{test}(\theta) = \frac{1}{2d_{tt}} \sum_{i=1}^{d_{tt}} \left(\theta^\top \phi(x^{(i)}) - y^{(i)}\right)^2$$

Variance Bias Trade-off

Error as a function of complexity

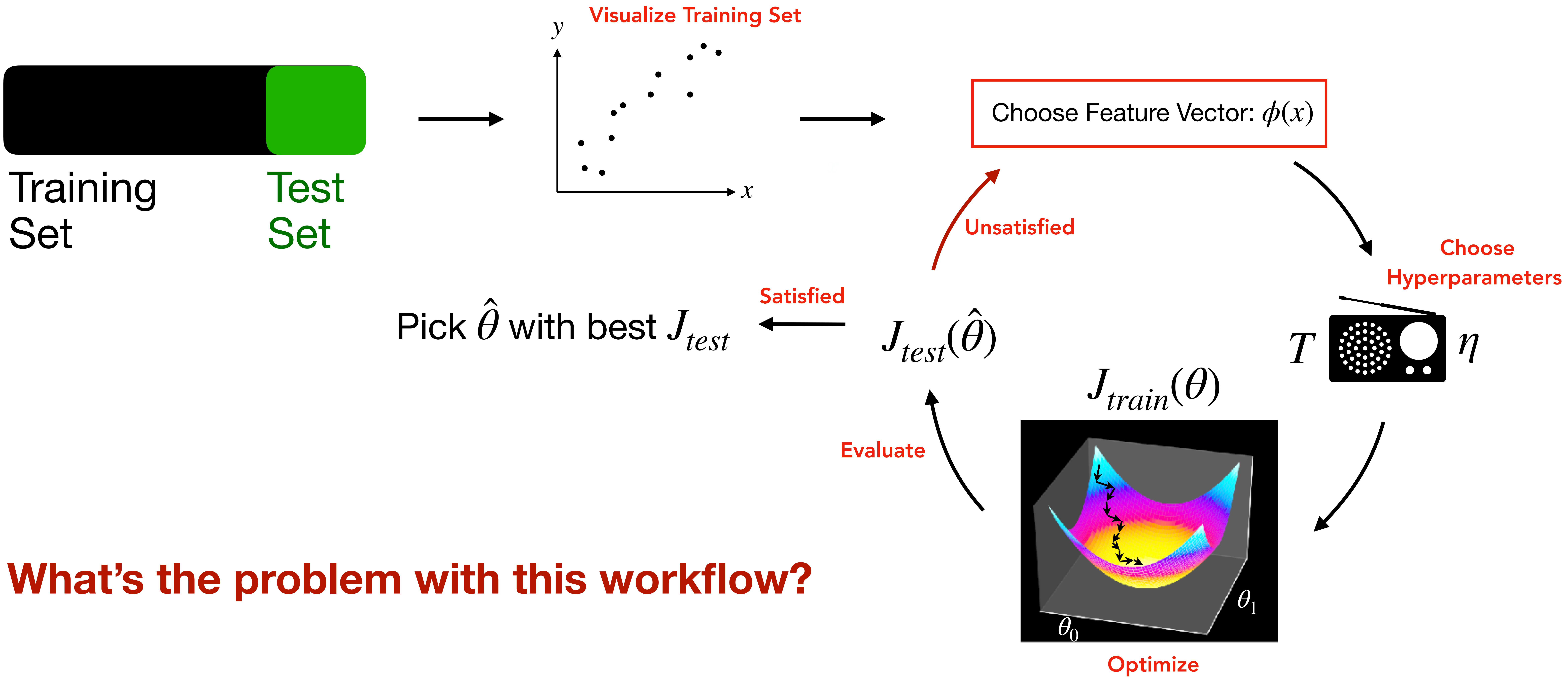


Other Hyperparameters

ϕ is not the only unknown parameter over which we want to optimize

- T: Number of Epochs
- η: Step size
- ϕ: Feature vector

Optimize over ϕ and other hyperparameters

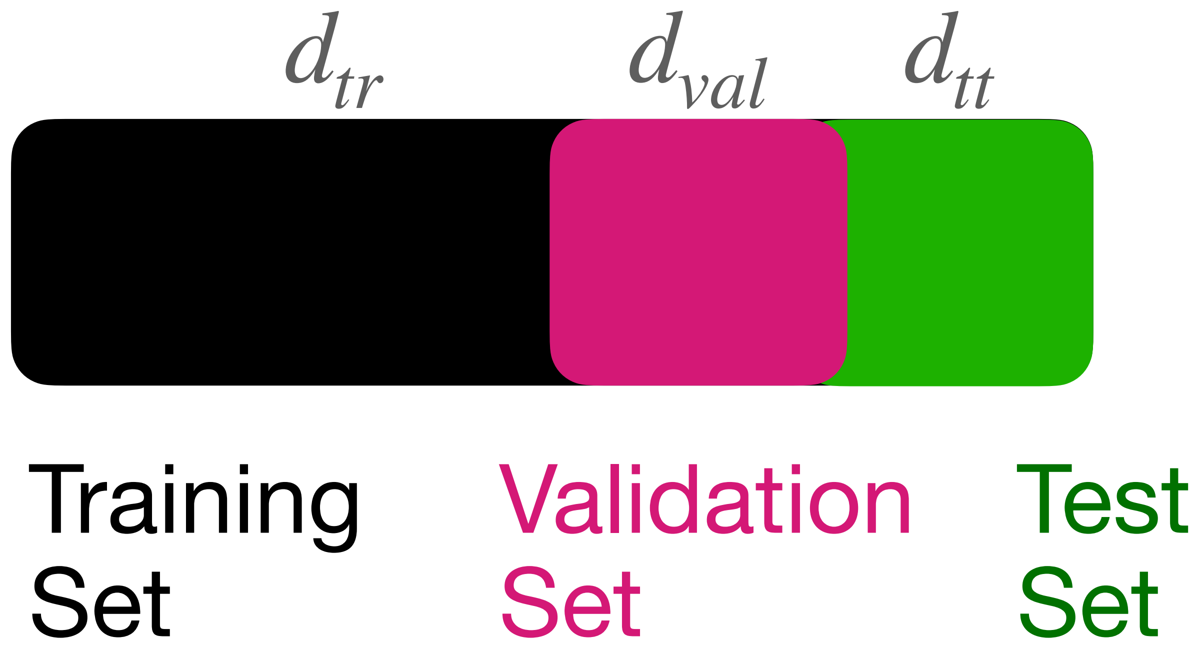


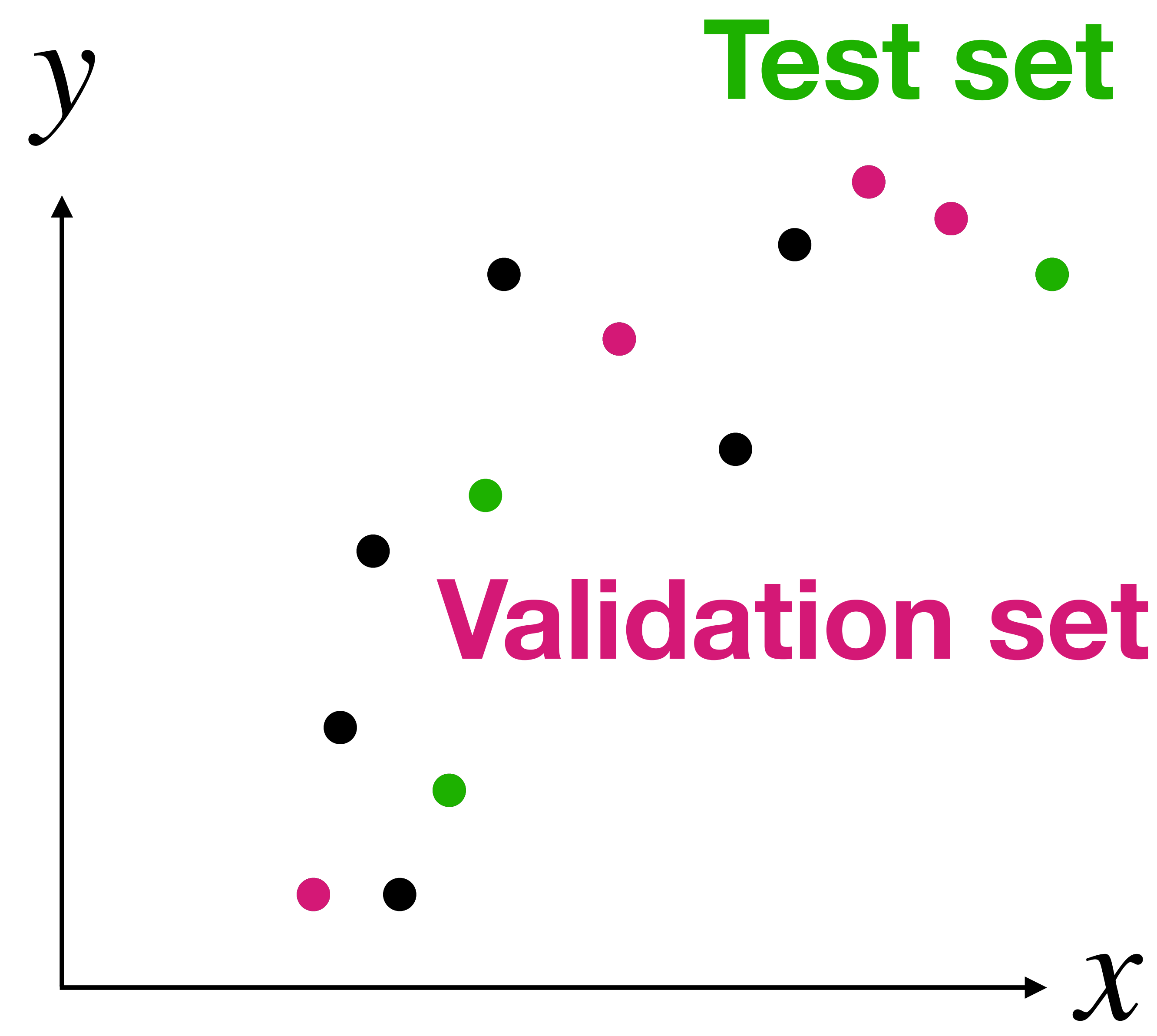
What's the problem with this workflow?

How do we tell that $\phi(\,\cdot\,)$ is good?

Define objective functions for each subset

Split data:





$$J_{train}(\theta) = \frac{1}{2d_{tr}} \sum_{i=1}^{d_{tr}} \left(\theta^\top \phi(x^{(i)}) - y^{(i)}\right)^2$$

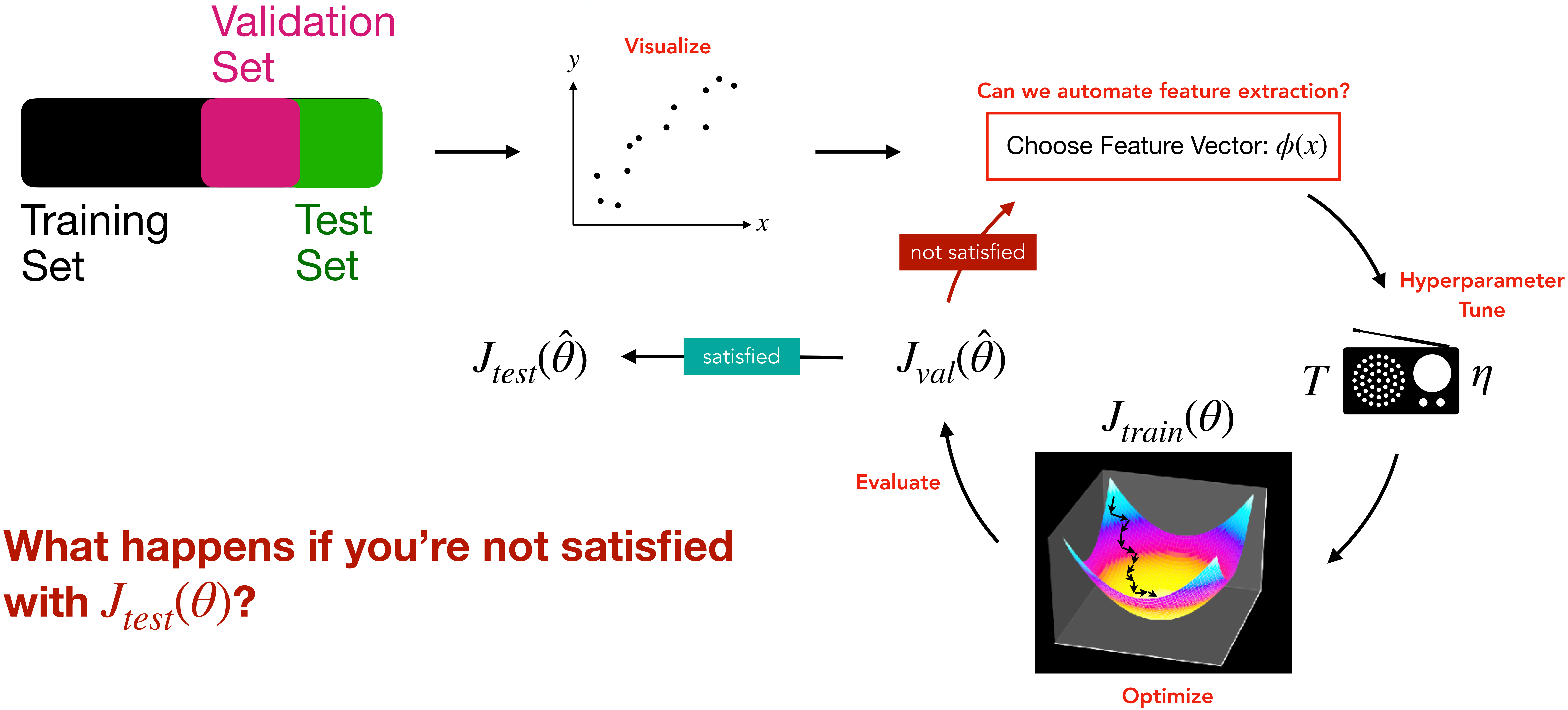
$$J_{val}(\theta) = \frac{1}{2d_{val}} \sum_{i=1}^{d_{val}} \left(\theta^\top \phi(x^{(i)}) - y^{(i)}\right)^2$$

$$J_{test}(\theta) = \frac{1}{2d_{tt}} \sum_{i=1}^{d_{tt}} \left(\theta^\top \phi(x^{(i)}) - y^{(i)}\right)^2$$

Test set: evaluate model **at the end** of hyperparameter optimization

Validation set: evaluation model **during** hyperparameter optimization

Machine Learning workflow - Cross Validation



What happens if you're not satisfied with $J_{test}(\theta)$?

Remedies to Overfitting

Practical tips to decrease overfitting

- Make the model simpler if it's overfitting, and more complex if it's underfitting
  - **Recursive Feature Elimination:** start with all features and drop them one by one while tracking the loss
  - Get rid of features that you think are irrelevant in predicting the desired output
- Add a regularization term that makes the hypothesis class smaller

$$J_{reg}(\theta) = J(\theta) + \lambda R(\theta)$$

Regularization

Force fitting parameters to be smaller - ‘shrink’ hypothesis class

$$h_\theta(x) = 100.2 + 50.6x + 70.4x^2 + 1345x^3 + 200.3x^4$$

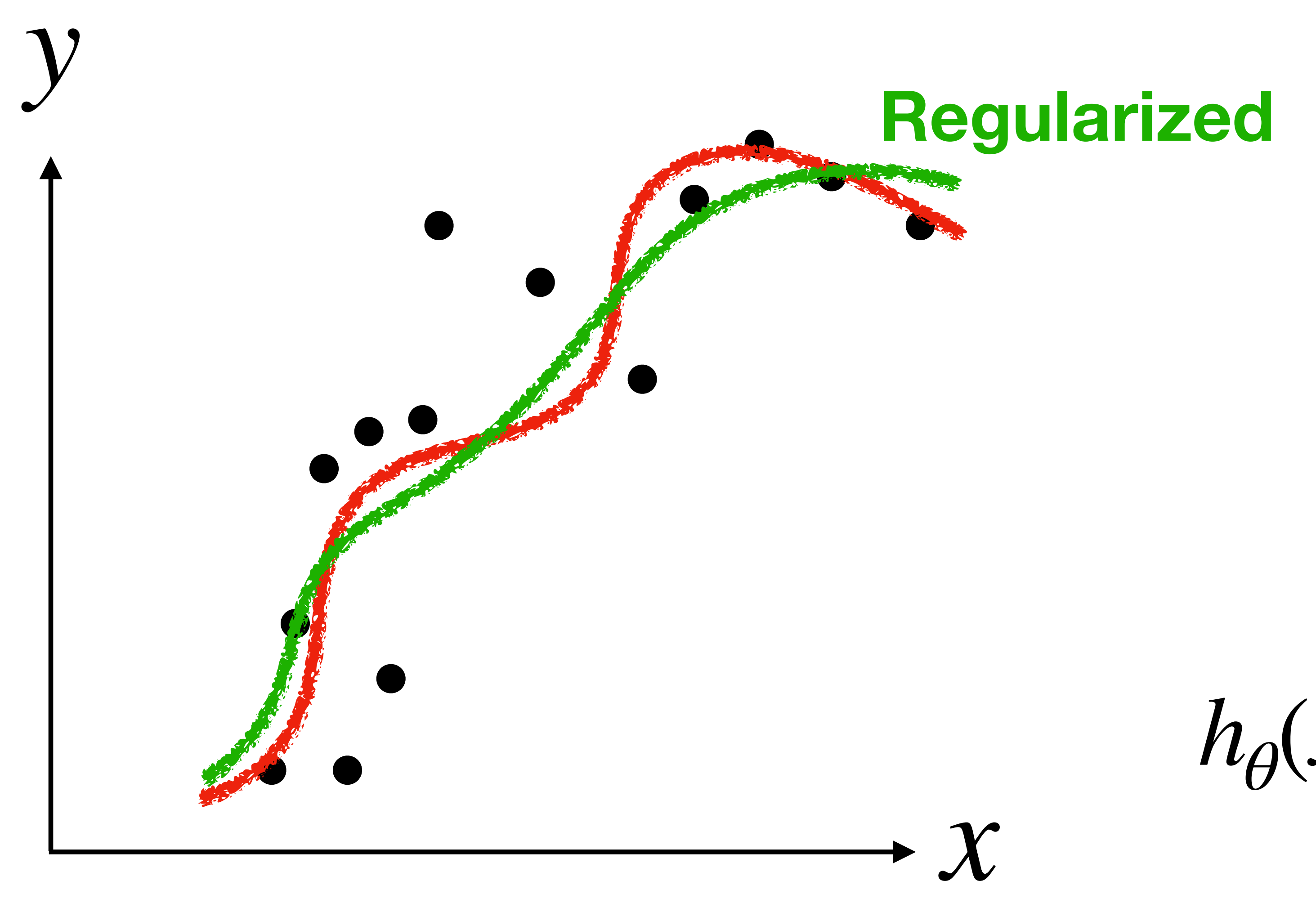
$$J_{reg}(\theta) = J(\theta) + \lambda\, R(\theta)$$

L1 Regularization

$$R(\theta) = \|\theta\|_1$$

$$h_\theta(x) = 5.1x + 7.2x^2 + 3.3x^4$$

Less coefficients



L2 Regularization

$$R(\theta) = \|\theta\|_2$$

$$h_\theta(x) = .1 + 5.2x + 7.4x^2 + .05x^3 + 2.3x^4$$

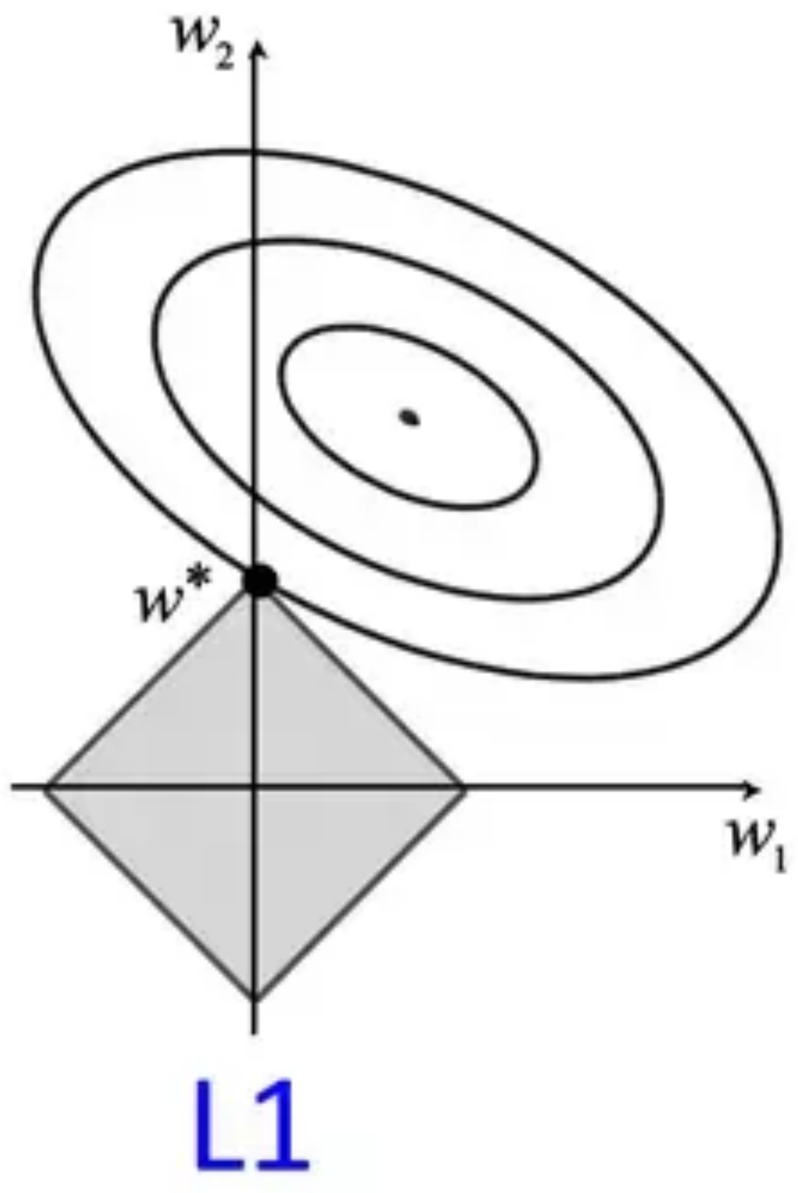
Smaller coefficients

Regularization

Force fitting parameters to be smaller - ‘shrink’ hypothesis class

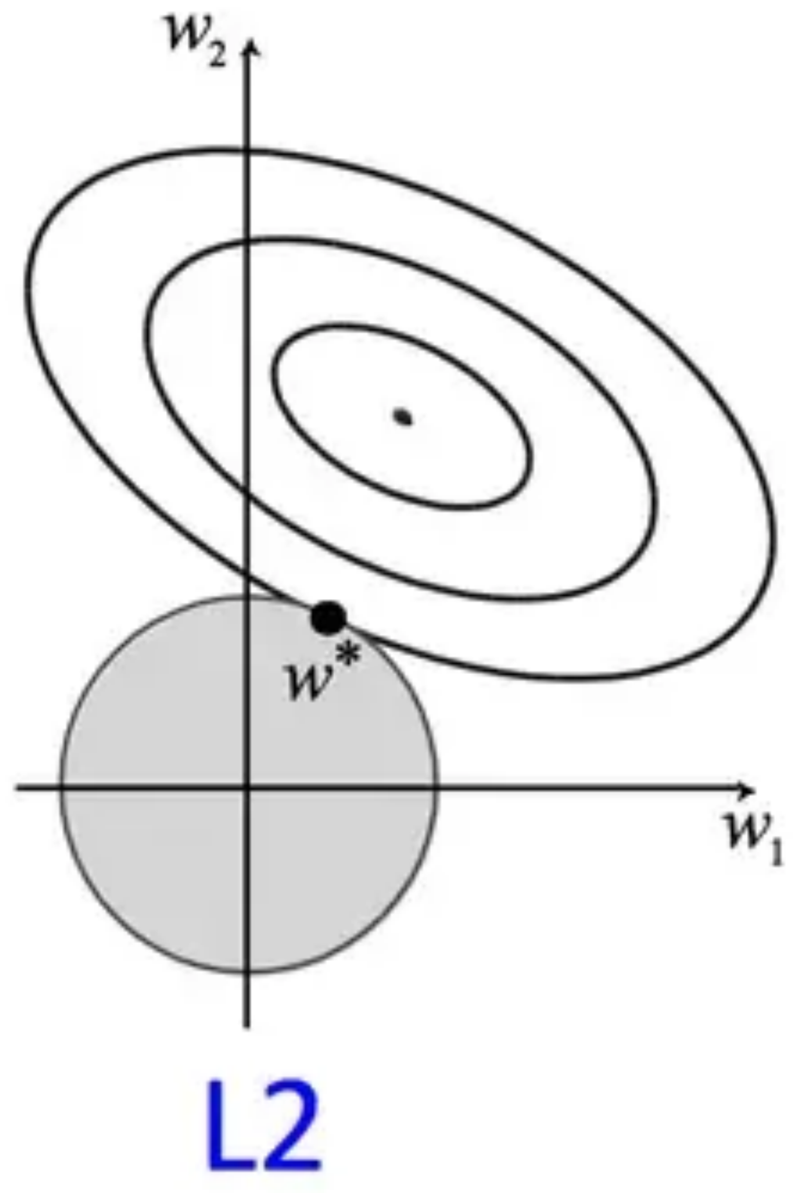
L1 Regularization

$$R(\theta) = \|\theta\|_1$$



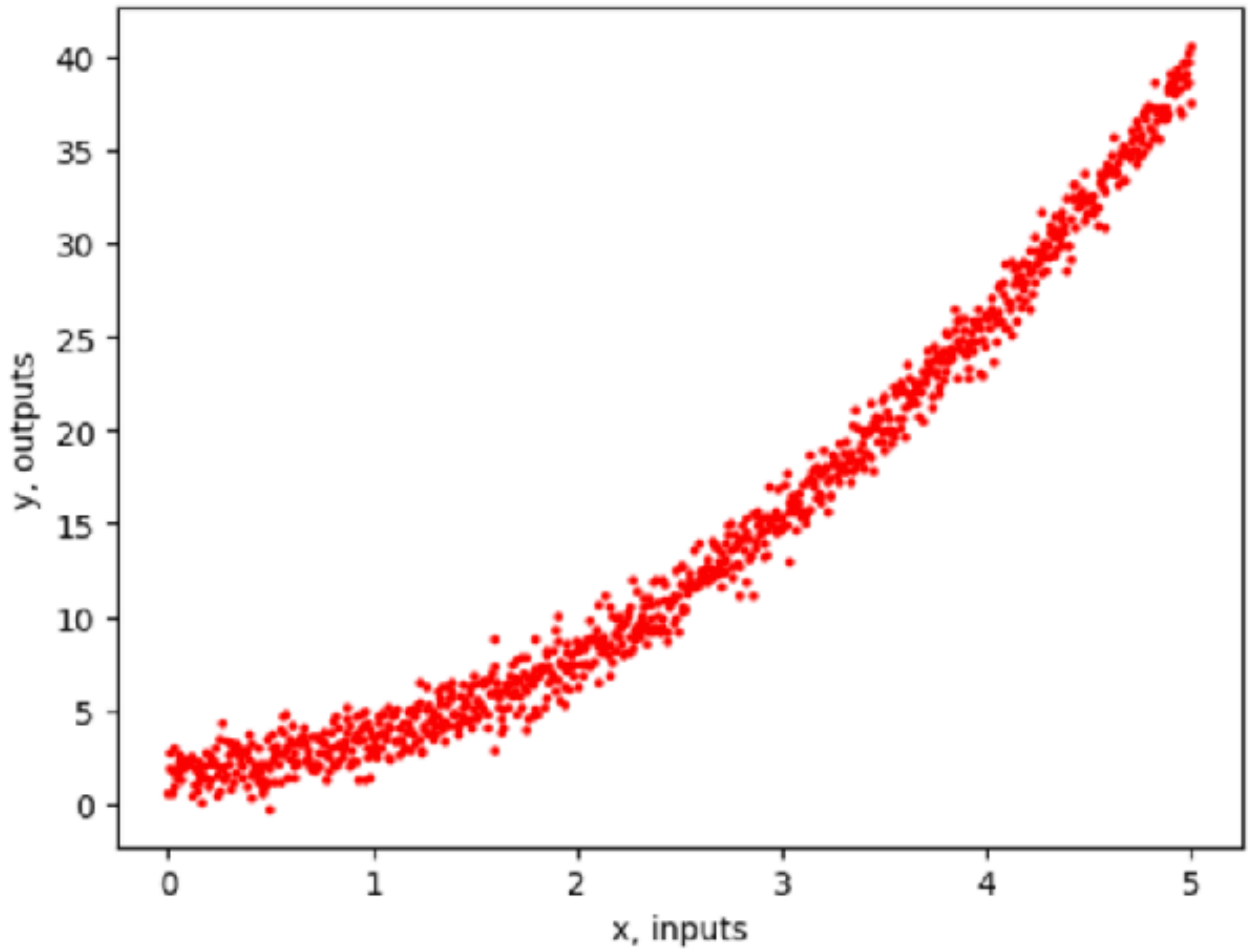
L2 Regularization

$$R(\theta) = \|\theta\|_2$$



Create synthetic data

```
# synthetic parameters
num_points = 1000
var = 1
a = 1.5
b = 2

# generate data
x = np.linspace(0, 5, num_points)
y = 1.5 * x**2 + b + var * np.random.normal(0, 1, num_points)
```



Feature engineering (design matrix)

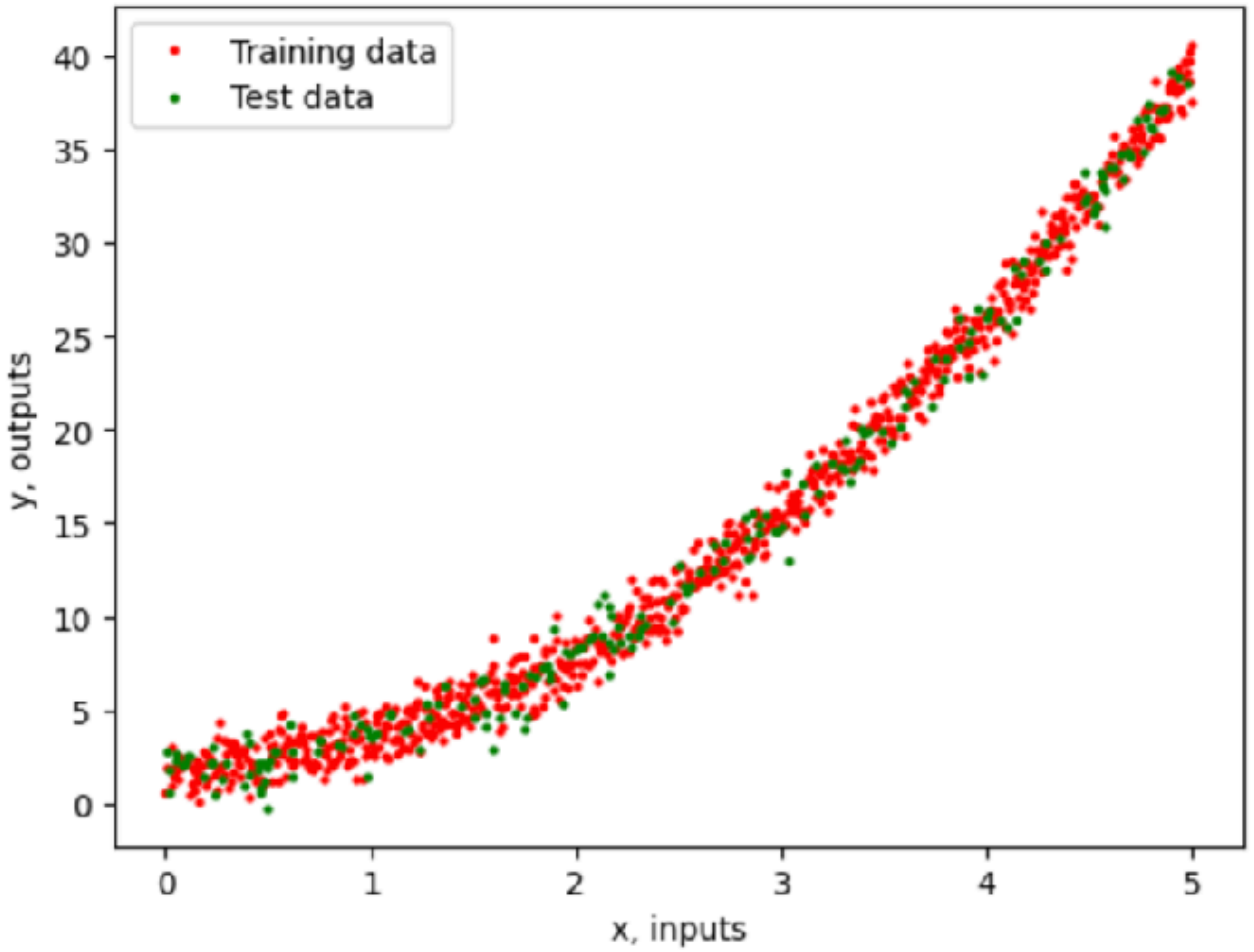
```python
# Create features

def design_matrix(x, degree):
    X = np.zeros((len(x), degree+1))
    for i in range(X.shape[1]):
        X[:, i] = x**i
    return X


degree = 2
X = design_matrix(x, degree)
y = y.reshape(-1, 1)
```

Shuffle and split

```
# Split data

n_train = int(0.8 * num_points)
n_test = num_points - n_train

shuff_index = np.random.permutation(num_points)
X_shuffle = X[shuff_index]
y_shuffle = y[shuff_index]

X_train = X_shuffle[:n_train]
X_test = X_shuffle[n_train:]
y_train = y_shuffle[:n_train]
y_test = y_shuffle[n_train:]
```

Visualize (training set)

```
# Plot training data
fig = plt.figure()
plt.plot(X_train[:, 1], y_train, 'ro', ms=2, label='Training data')
plt.plot(X_test[:, 1], y_test, 'go', ms=2, label='Test data')
plt.xlabel('x, inputs')
plt.ylabel('y, outputs')
plt.legend()
plt.show()
```



Define cost function and Gradient Descent

Cost function

```
def cost_function(X, y, theta):
    m = len(y)
    return 1/(2*m) * np.sum((X @ theta - y)**2)
```

Gradient Descent Function

```
def gradient_descent(X, y, theta, learning_rate, num_iters):
    m = len(y)
    J_history = np.zeros(num_iters)
    for i in range(num_iters):
        theta = theta - (learning_rate/m) * X.T @ (X @ theta - y)
        J_history[i] = cost_function(X, y, theta)
    return theta, J_history
```

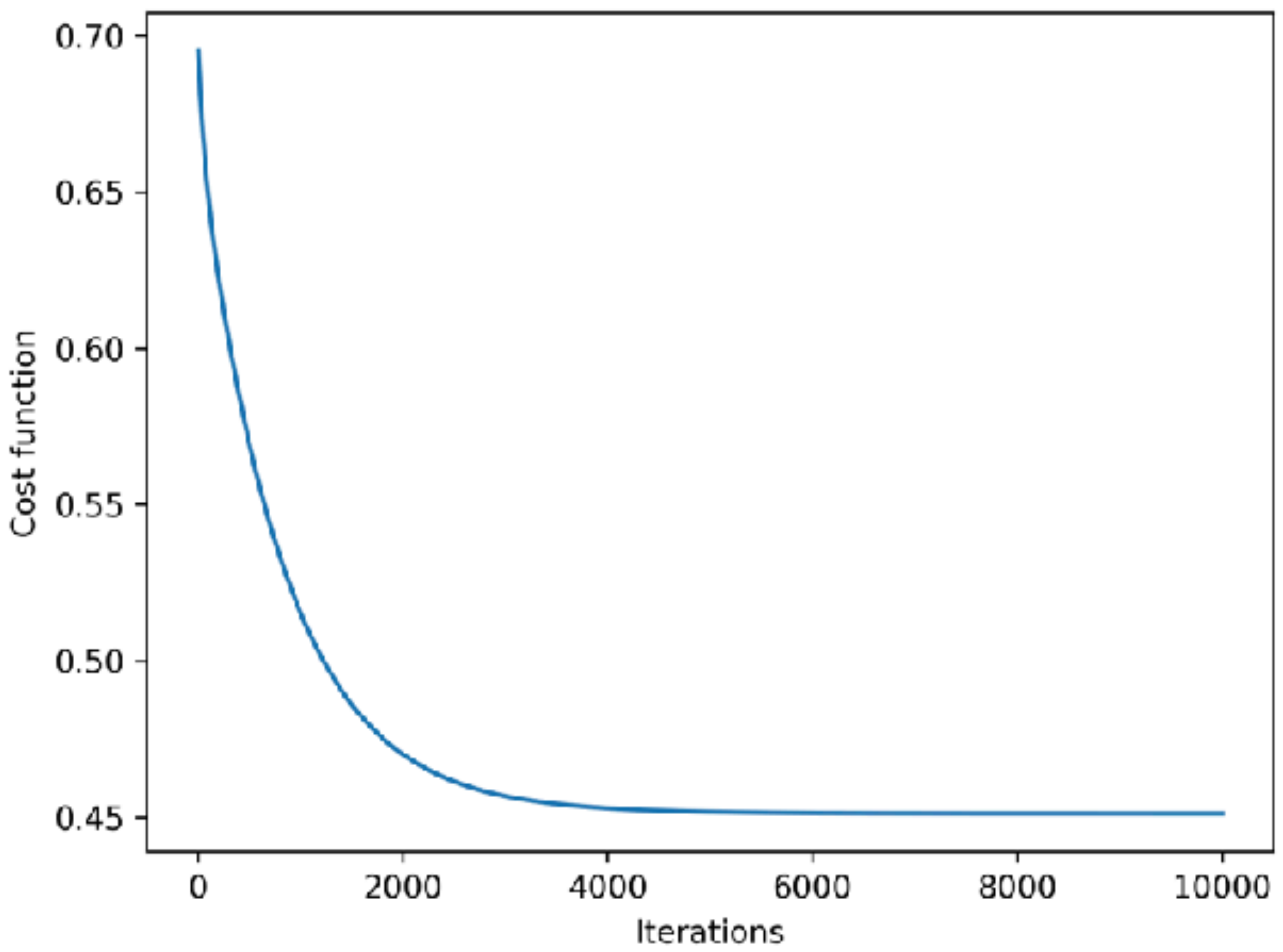
Gradient Descent Update

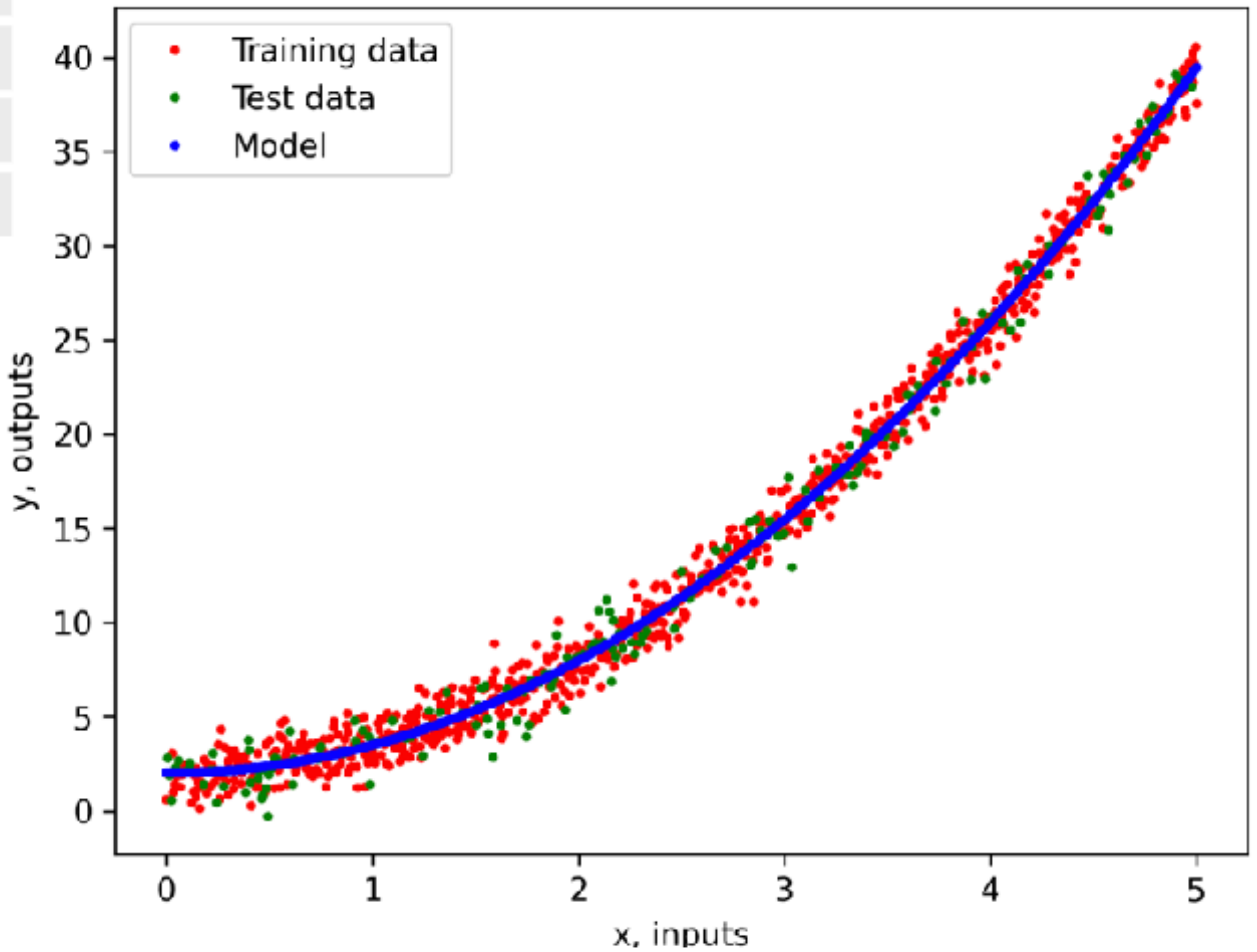
```
theta = np.random.randn(degree+1, 1)
learning_rate = 0.01
num_iters = 10000
theta, J_history = gradient_descent(X_train, y_train, theta, learning_rate, num_iters)
```

```
theta
✓ 0.0s

array([[ 2.04211351],
       [-0.02468485],
       [ 1.50370819]])
```

Plot result

```python
# plot comparison
fig = plt.figure()
plt.plot(X_train[:, 1], y_train, 'ro', ms=2, label='Training data')
plt.plot(X_test[:, 1], y_test, 'go', ms=2, label='Test data')
plt.plot(X_train[:, 1], X_train @ theta, 'bo', ms=2, label='Model')
plt.xlabel('x, inputs')
plt.ylabel('y, outputs')
plt.legend()
plt.show()
```





Plot result

```
# plot comparison
fig = plt.figure()
plt.plot(X_train[:, 1], y_train, 'ro', ms=2, label='Training data')
plt.plot(X_test[:, 1], y_test, 'go', ms=2, label='Test data')
plt.plot(X_train[:, 1], X_train @ theta, 'bo', ms=2, label='Model')
plt.xlabel('x, inputs')
plt.ylabel('y, outputs')
plt.legend()
plt.show()
```

Train and Test loss Comparison

```
test_loss = cost_function(X_test, y_test, theta)
train_loss = cost_function(X_train, y_train, theta)

print(f'Test loss: {test_loss}')
print(f'Train loss: {train_loss}')
```

✓ 0.0s

```
Test loss: 0.508970692715051
Train loss: 0.4511700483353495
```