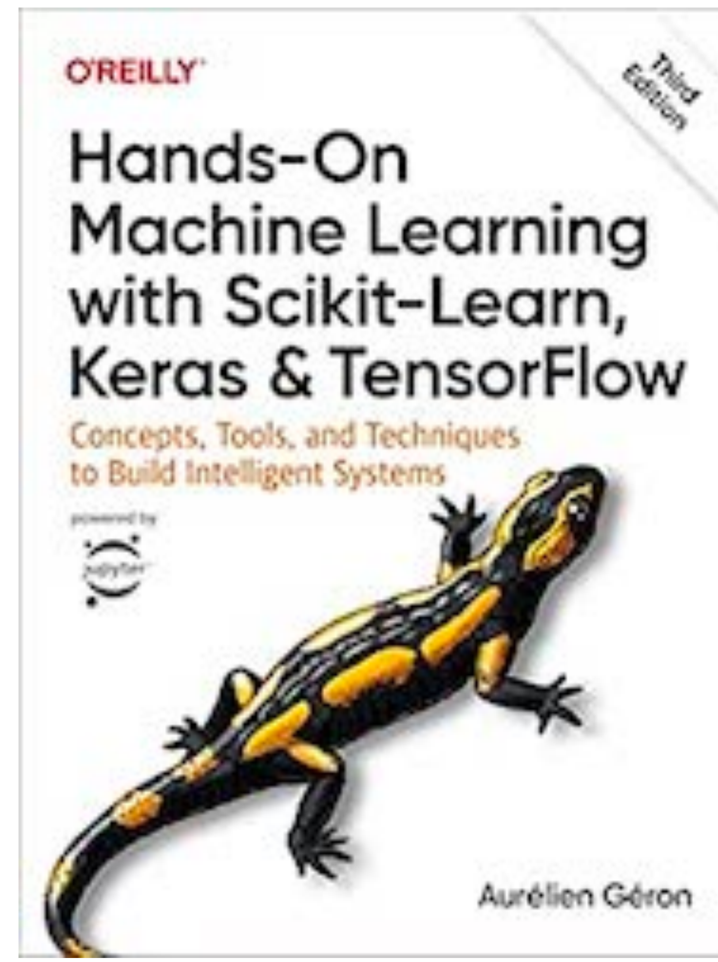
Machine Learning Security

4 Training Models



Made Aug 26, 2023

Topics

- Linear Regression
- Gradient Descent
- Polynomial Regression
- Learning Curves
- Regularized Linear Models
- Logistic Regression

Linear Regression

Two Ways to Train

- Closed-form equation
  - Directly compute parameters for best fit
- Gradient Descent (GD)
  - Iterative process
  - Gradually tweak parameters to minimize the cost function
  - Types of gradient descent
    - Batch GD
    - Mini-batch GD
    - Stochastic GD

Linear Regression

$$life\_satisfaction = \theta_0 + \theta_1 \times GDP\_per\_capita$$

Equation 4-1. Linear regression model prediction

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- The prediction depends linearly on the inputs (x_i)

Cost Function

Equation 4-3. MSE cost function for a linear regression model

$$\mathrm{MSE}\left(\mathbf{X}, h_{\boldsymbol{\theta}}\right) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}^{(i)} - y^{(i)}\right)^{2}$$

- Mean Squared Error

The Normal Equation

Equation 4-4. Normal equation

$$\widehat{\theta} = \left(\mathbf{X}^\intercal \mathbf{X}\right)^{-1} \mathbf{X}^\intercal \mathbf{y}$$

In this equation:

- **$\widehat{\theta}$ is the value of θ that minimizes the cost function.**
- **y is the vector of target values containing *y*(1) to *y*(*m*).**

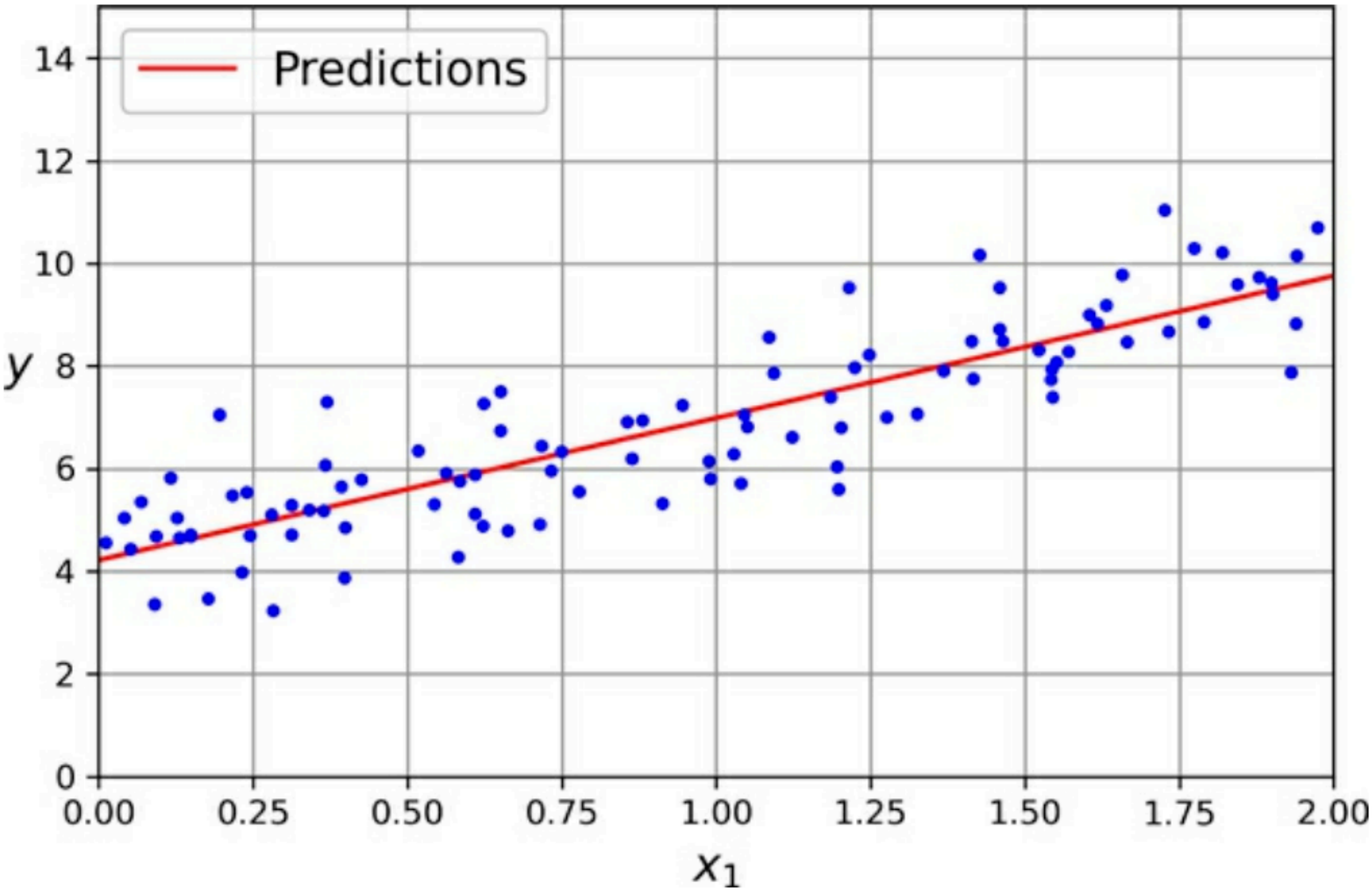


Figure 4-2. Linear regression model predictions

Computational Complexity

- Computes the matrix inverse of X^TX
  - $(\boldsymbol{n} + 1) \times (\boldsymbol{n} + 1)$ for $\boldsymbol{n}$ features
- Computational complexity is $\boldsymbol{O}(\boldsymbol{n}^{2.4})$ to $\boldsymbol{O}(\boldsymbol{n}^3)$
  - Doubling $\boldsymbol{n}$ increases computation time by a factor of 5 to 8
- Once the model is trained, prediction is fast
  - Complexity is linear in number of predictions and number of features

When the Closed-form Equation Method Fails

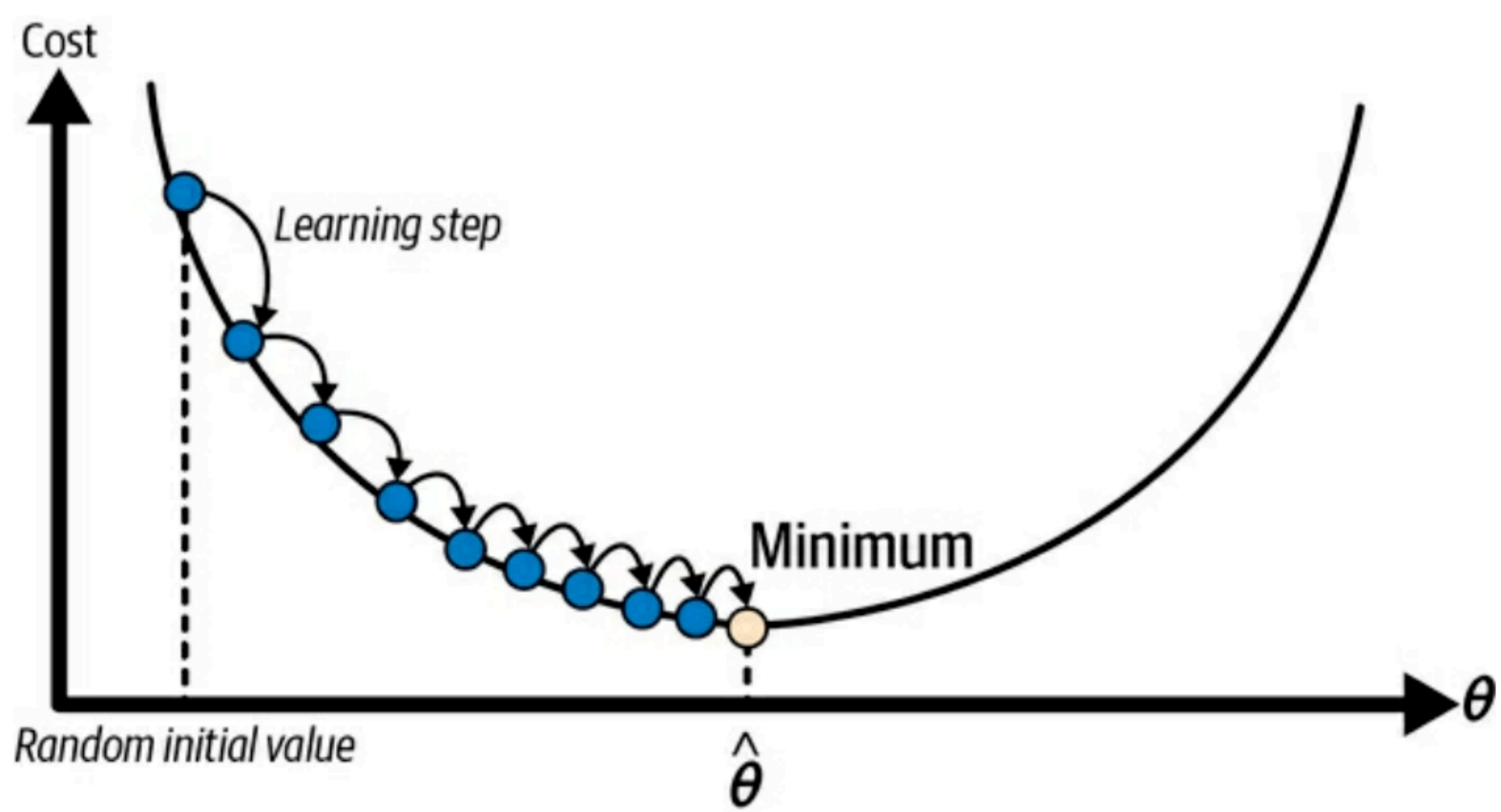
- There are a large number of features
- There are too many training instances to fit in memory

- For these cases, use Gradient Descent

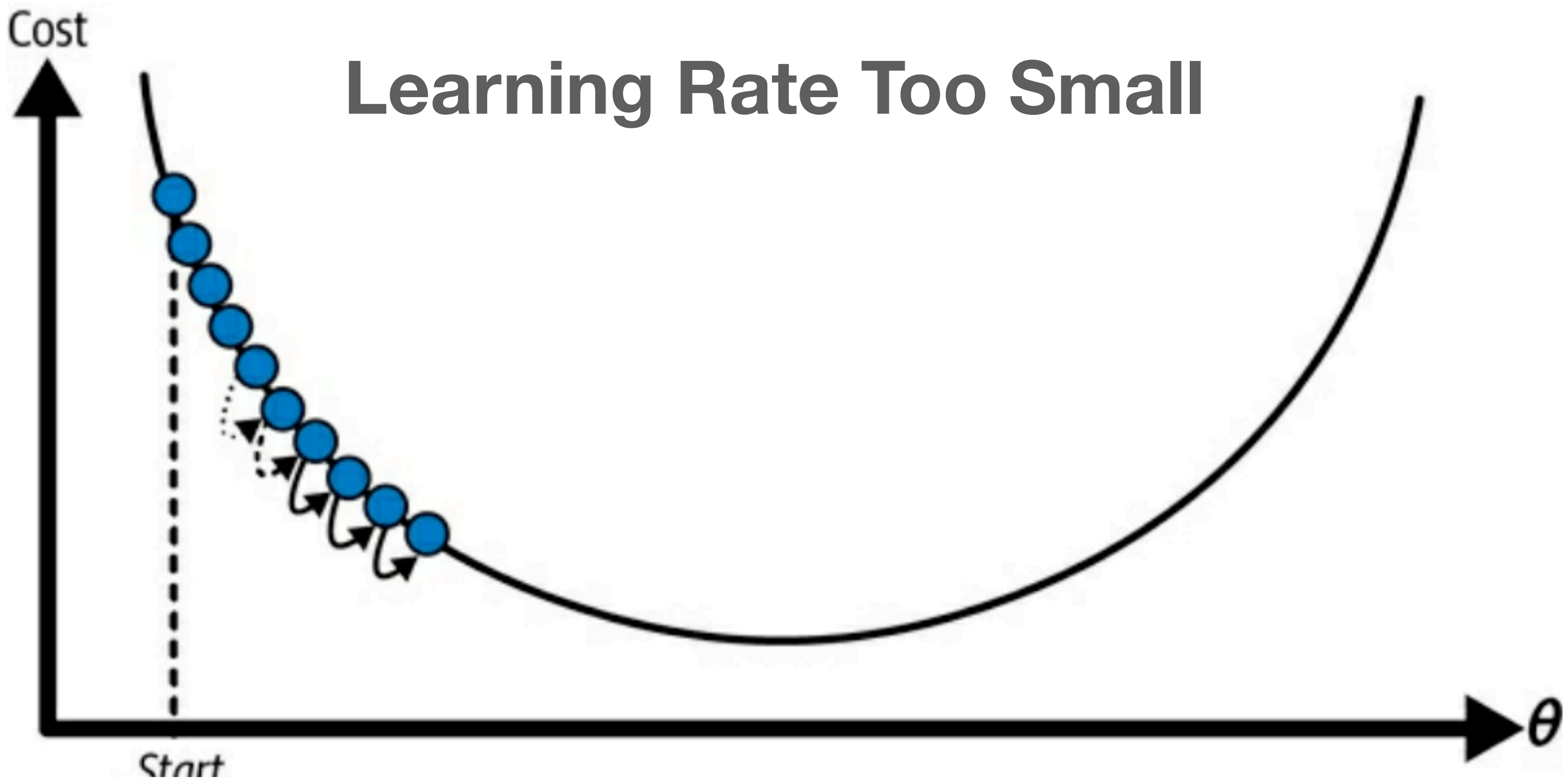
Gradient Descent

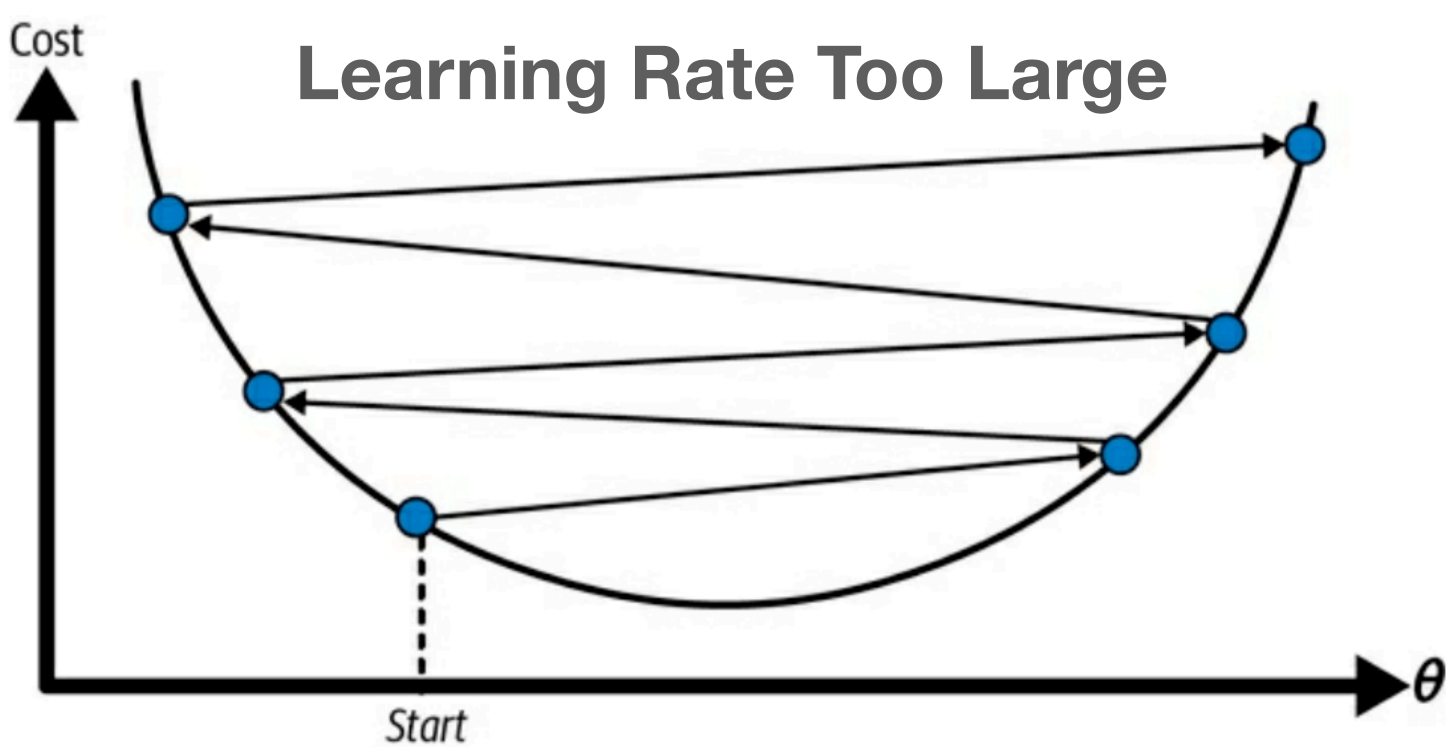
How Gradient Descent Works

- Start with random parameters
- Step in direction of steepest descent



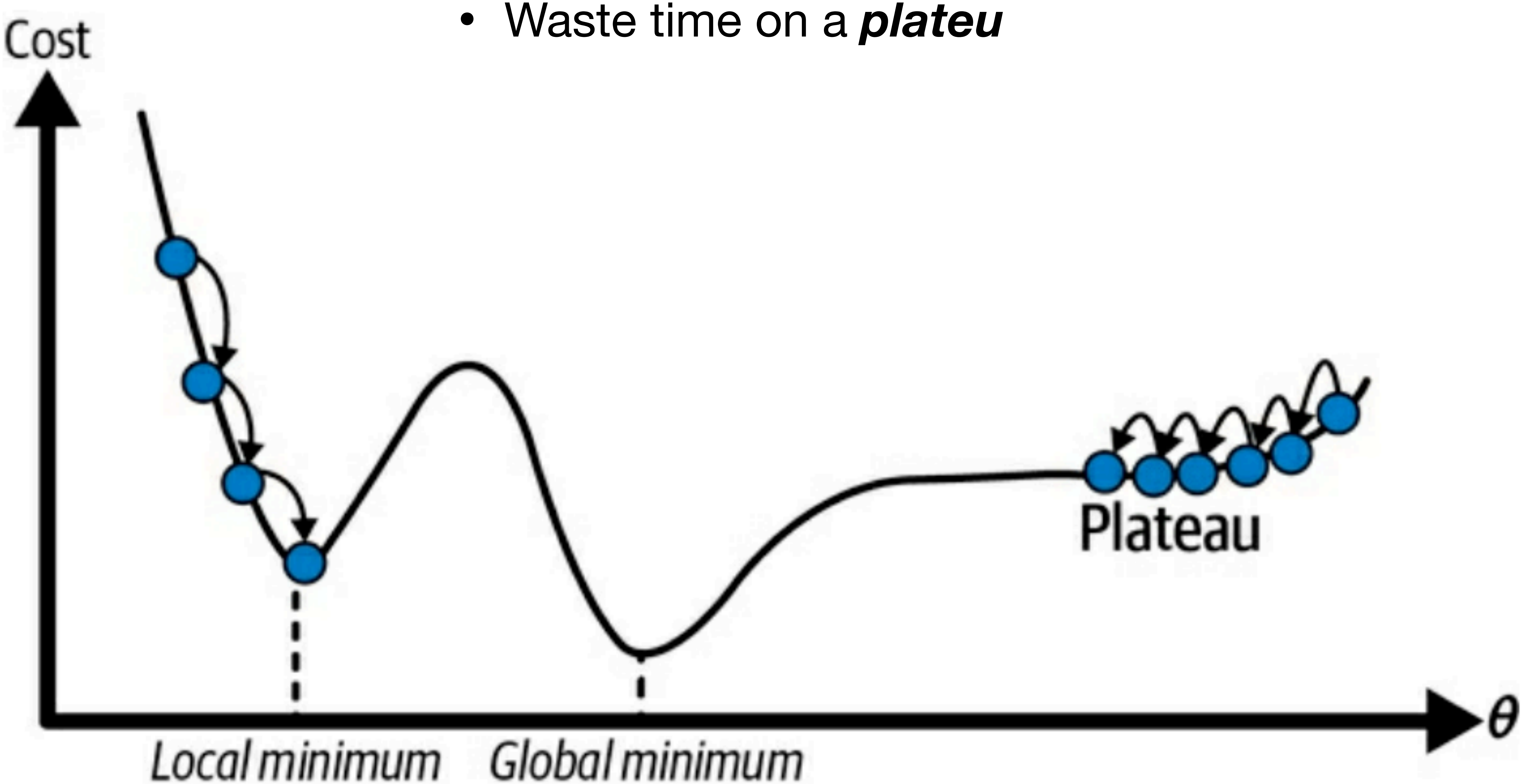
Learning Rate





Pitfalls

- Converge to a ***local minimum***
- Waste time on a ***plateu***

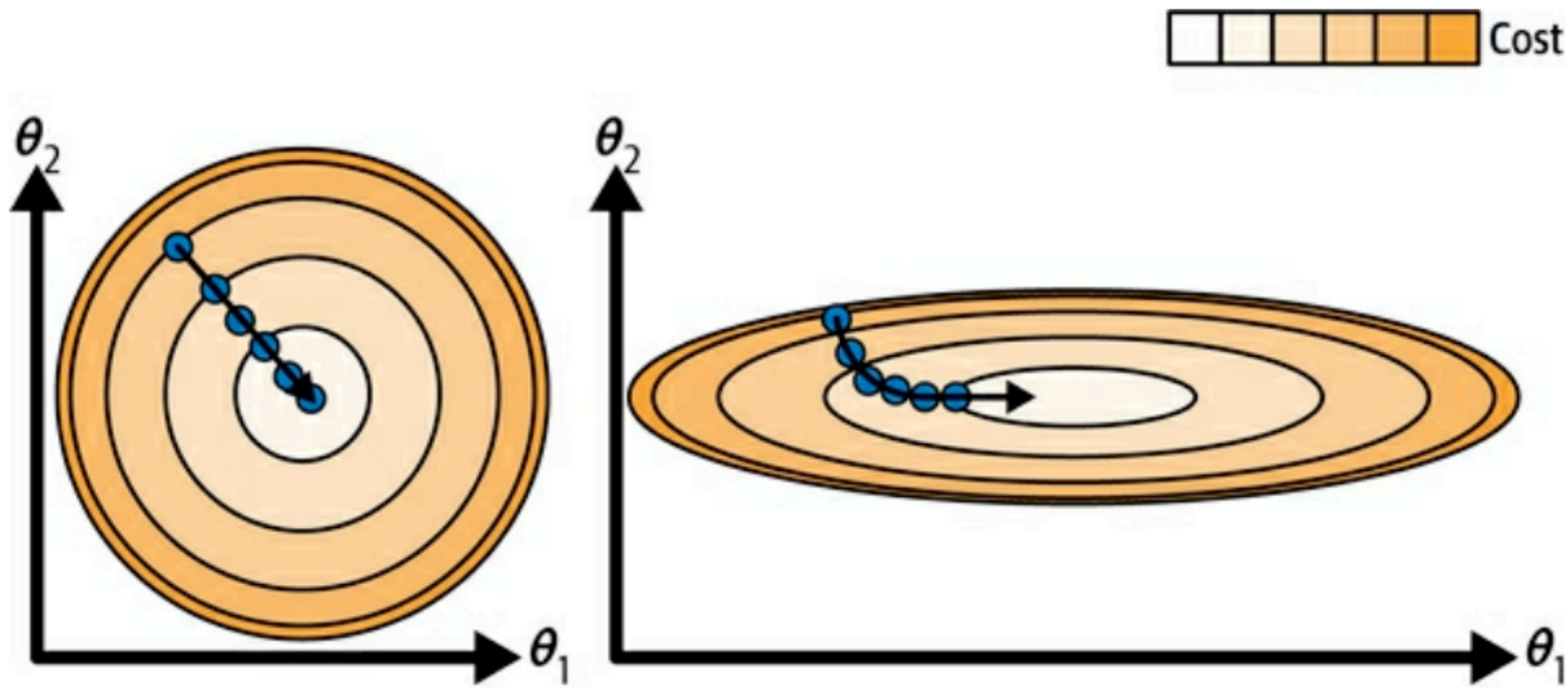


Mean Squared Error

- A ***convex function***
  - No local minima, just one global minimum
- Continuous function
  - Slope never changes abruptly

Feature Scaling

- Converges most rapidly when all features have the same scale



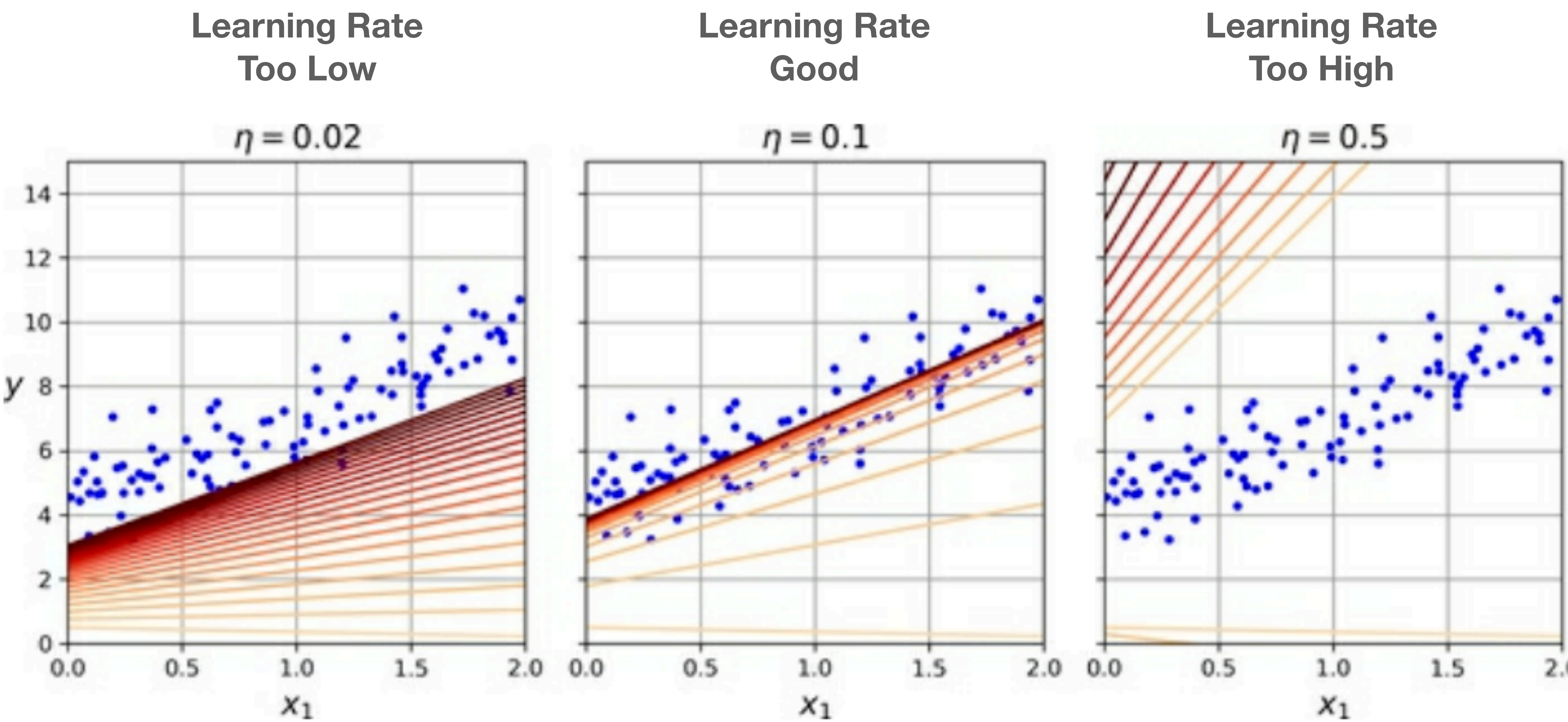
Batch Gradient Descent

Equation 4-5. Partial derivatives of the cost function

$$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^\intercal \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

- The slope of the descent can be easily computed for linear models
- It uses the whole training data at each step
  - Very slow on large training sets
- Scales linearly with number of features
  - Much faster than Normal equation

Gradient Descent with Various Learning Rates



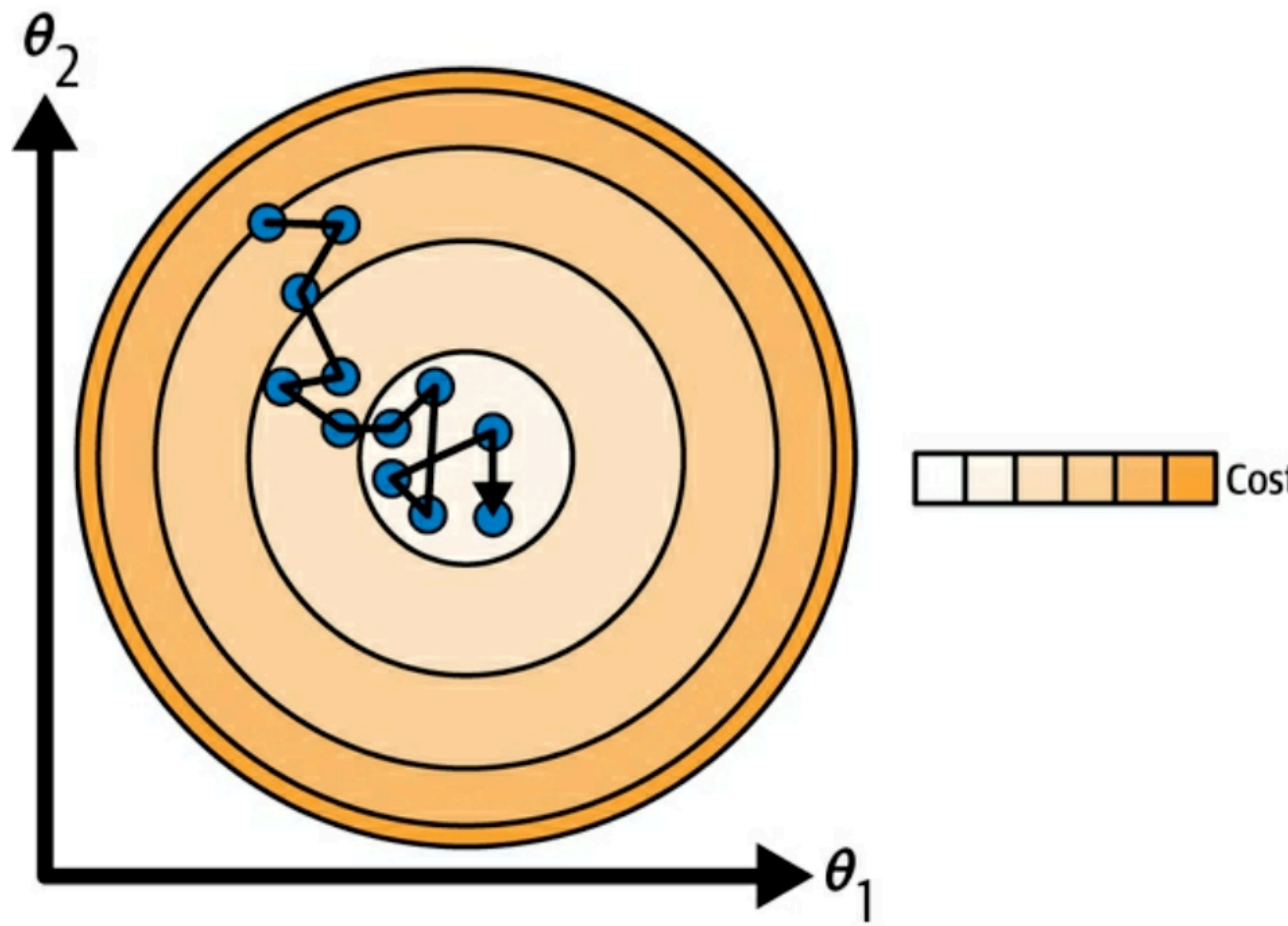
- Find the best learning rate with a grid search

Stochastic Gradient Descent (SGD)

- Gradient descent uses the whole training set for each step
- ***Stochastic Gradient Descent***
  - Picks a random instance in the training set at every step
  - Computes the gradient from that instance
  - Much faster, and can use huge training sets

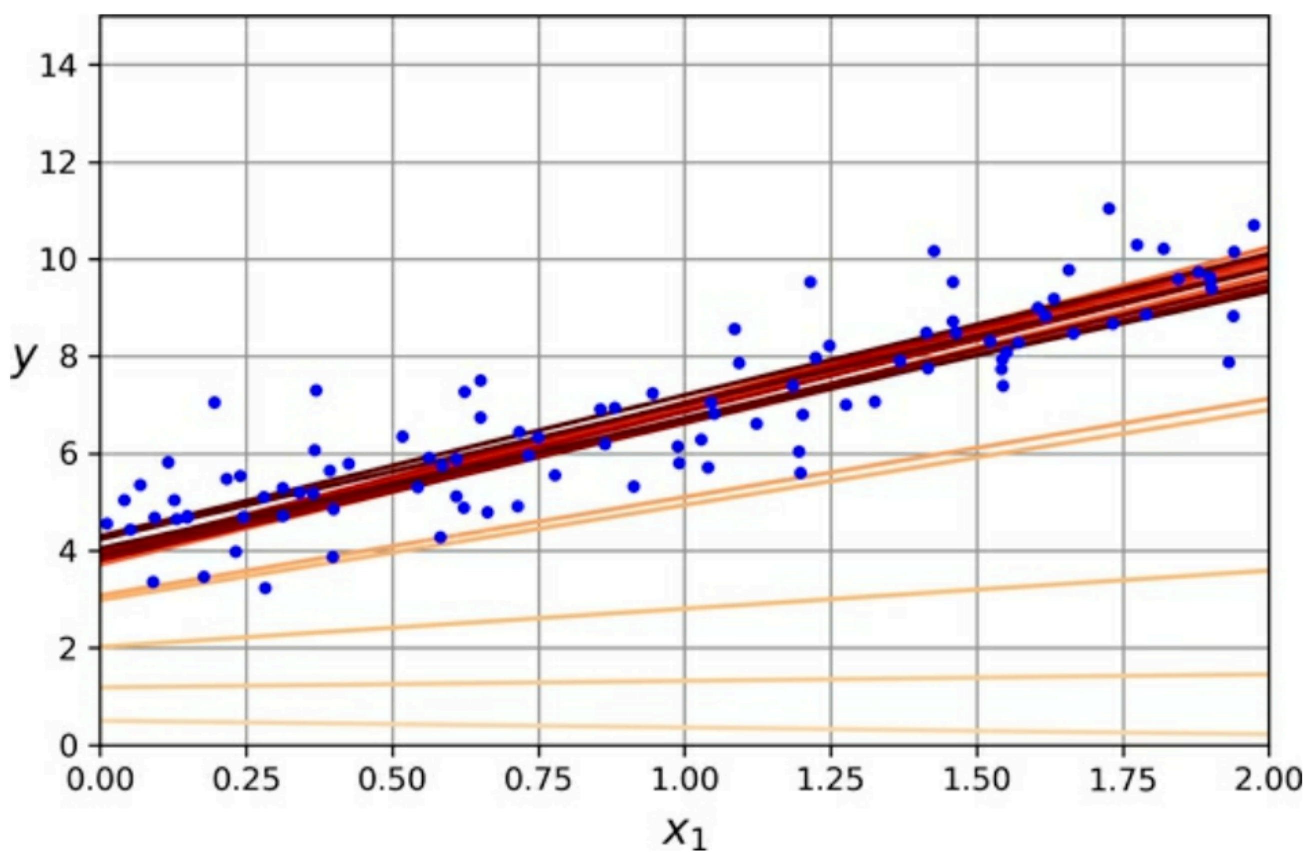
Stochastic Gradient Descent

- Cost does not decrease with each step
- Bounces around randomly
- Decreases on average
- Never settles down to minimum
- If cost function is irregular, this can help it jump out of local minima



Learning Schedule

- Gradually decrease learning rate
  - Causes stochastic gradient descent to settle at the global minimum
  - Similar to ***simulated annealing***

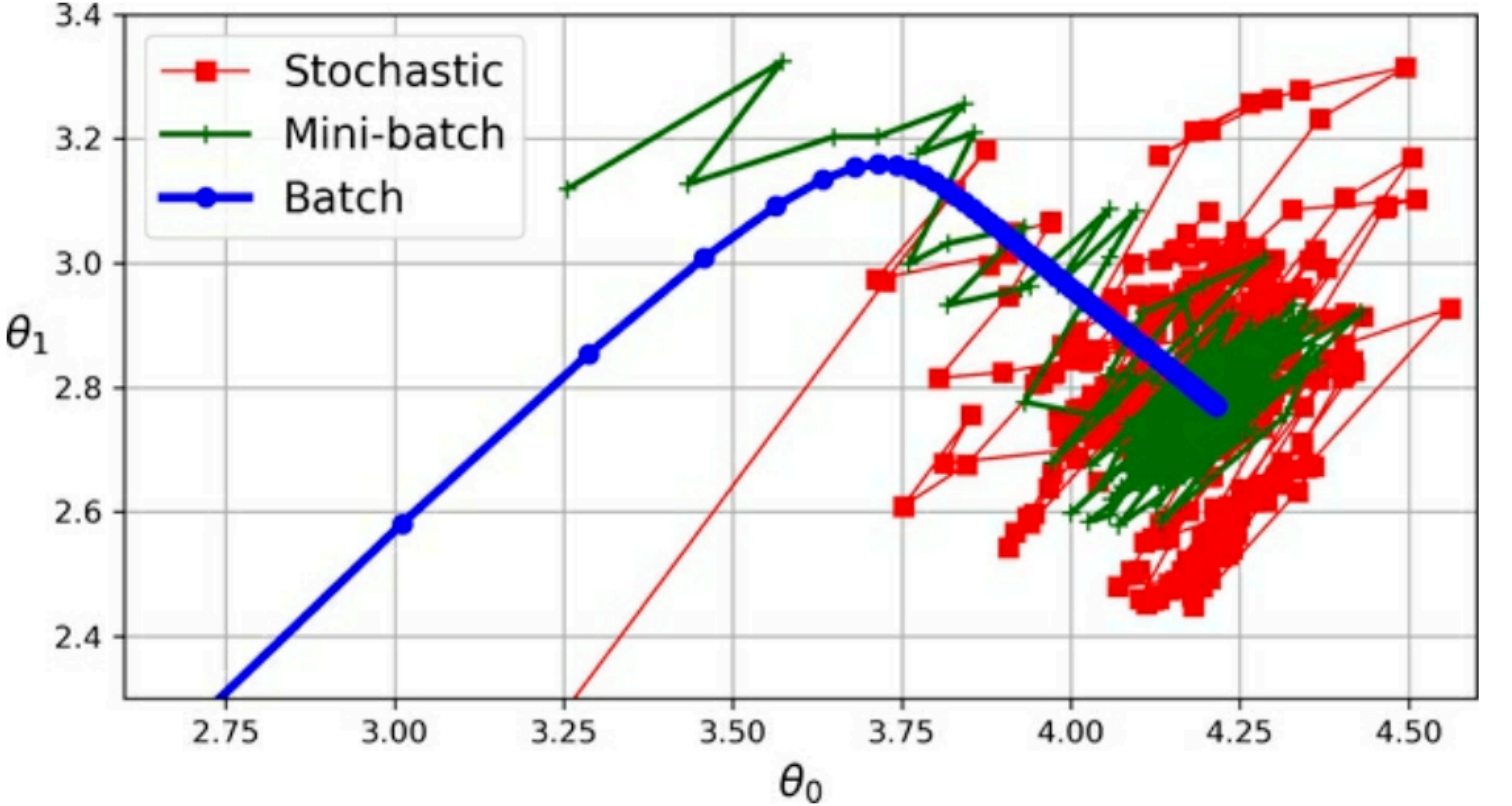


Shuffling

- If the training set is sorted
  - SGD will first use one category of data, then switch to another
  - It won't converge to the global minimum
- To avoid this, shuffle the training data at each epoch
  - Or pick each instance randomly

Mini-Batch Gradient Descent

- At each step, compute the gradient using small random sets of training instances (called ***mini-batches***)
- Gets a performance boost from hardware optimization of matrix operatins, especially GPUs
- Less erratic than SGD
  - Especially with large mini-batches
  - But it may get stuck in local minima


Stochastic
Mini-batch
Batch
3.4
3.2
3.0
2.8
2.6
2.4
θ_1
2.75
3.00
3.25
3.50
3.75
4.00
4.25
4.50
θ_0

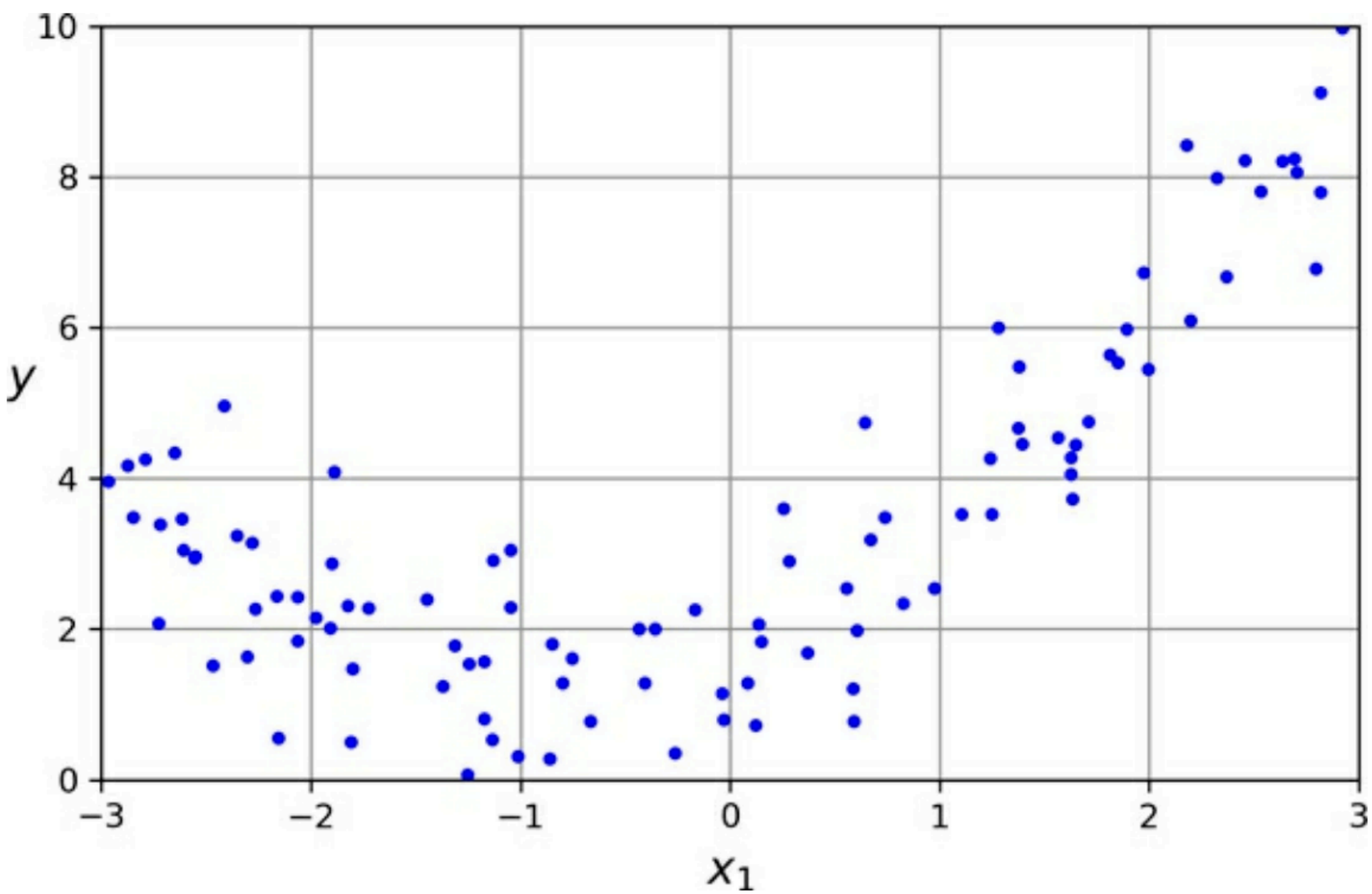
Polynomial Regression

Use a Linear Model to Fit Nonlinear Data

- Add powers of features as new features
- Train a linear model on the new features

Quadratic Data

```
np.random.seed(42)
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```



Add Squared Feature as a New Feature

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly_features = PolynomialFeatures(degree=2, include_bias=False)
>>> X_poly = poly_features.fit_transform(X)
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929,  0.56664654])
```

- Original X has only one value
- X_poly has two values per instance

Fit Linear Regression to Extended Training Data

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```

Not bad: the model estimates $\hat{y} = 0.56{x_1}^2 + 0.93x_1 + 1.78$ **when in fact the original function was** $y = 0.5{x_1}^2 + 1.0x_1 + 2.0 + \text{Gaussian noise}$**.**

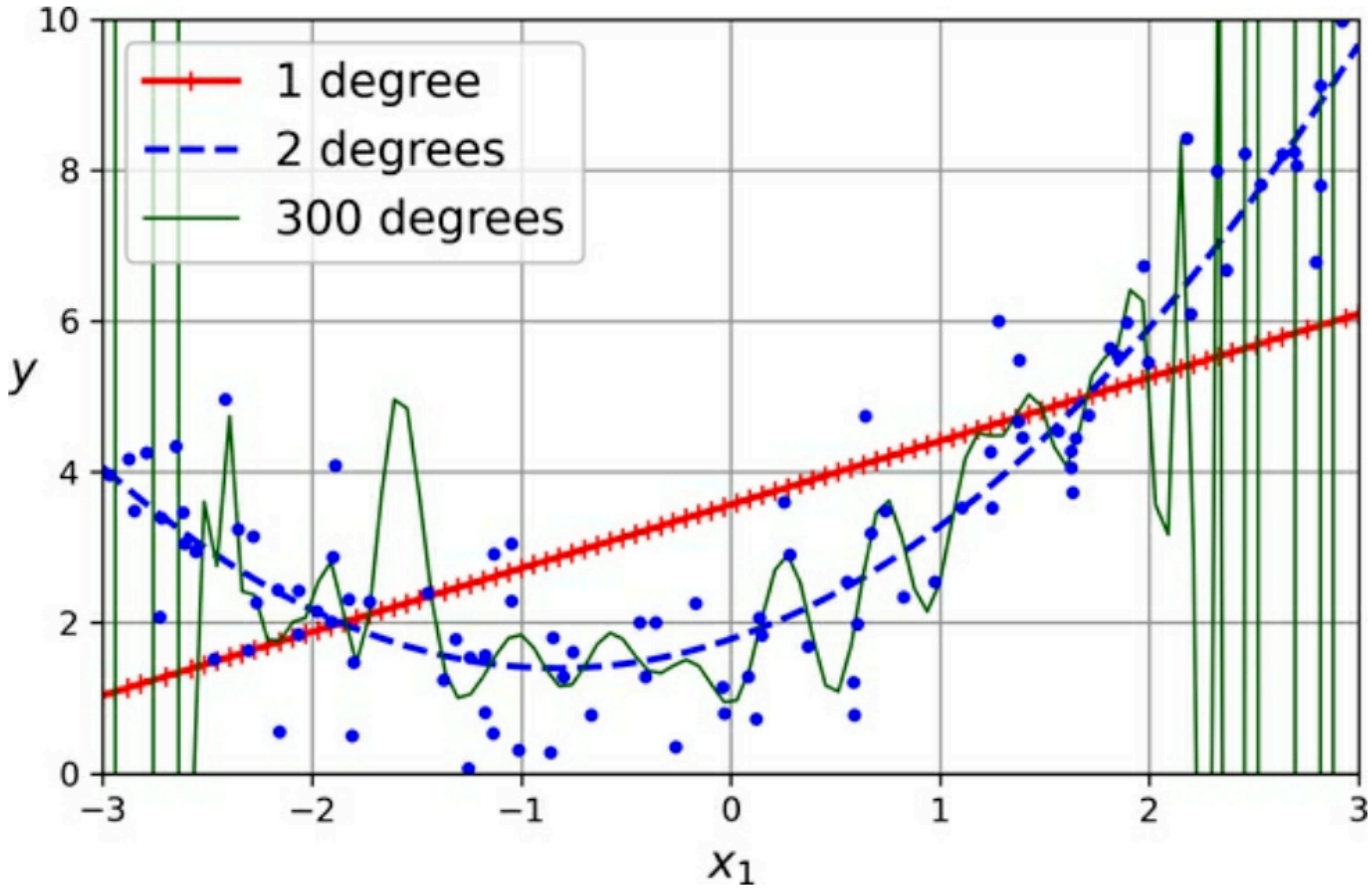
Kahoot!

Ch 4a

Learning Curves

Various Polynomials

- Line underfits the data (1 degree)
- 300 degrees overfits the data
- We could evaluate these models with cross-validation

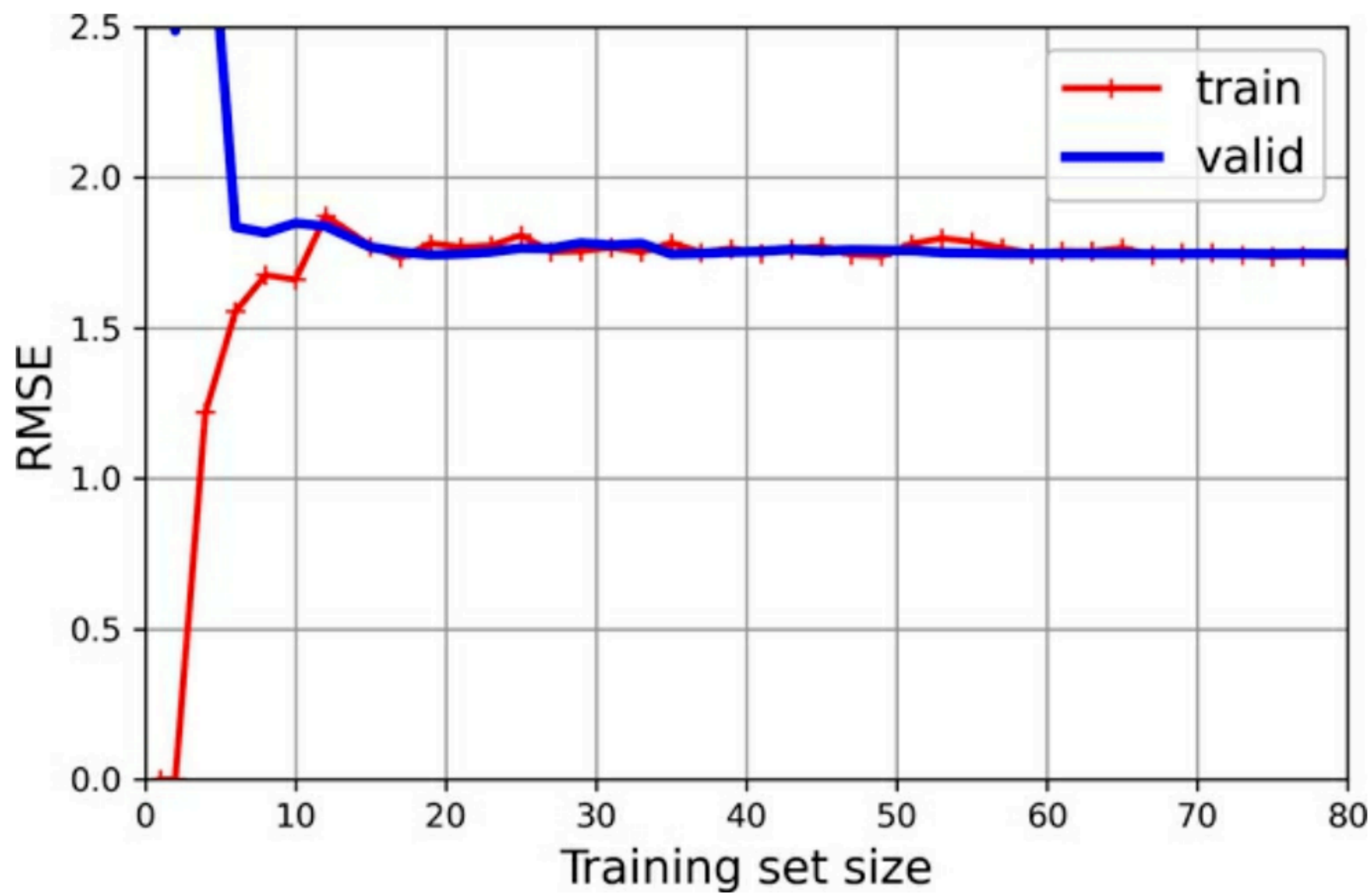


Learning Curves

- Another way to measure a model's performance
- Plot training error and validation error
  - As a function of training iteration
- If a model canot be trained incrementally, use gradually larger subsets of the training data

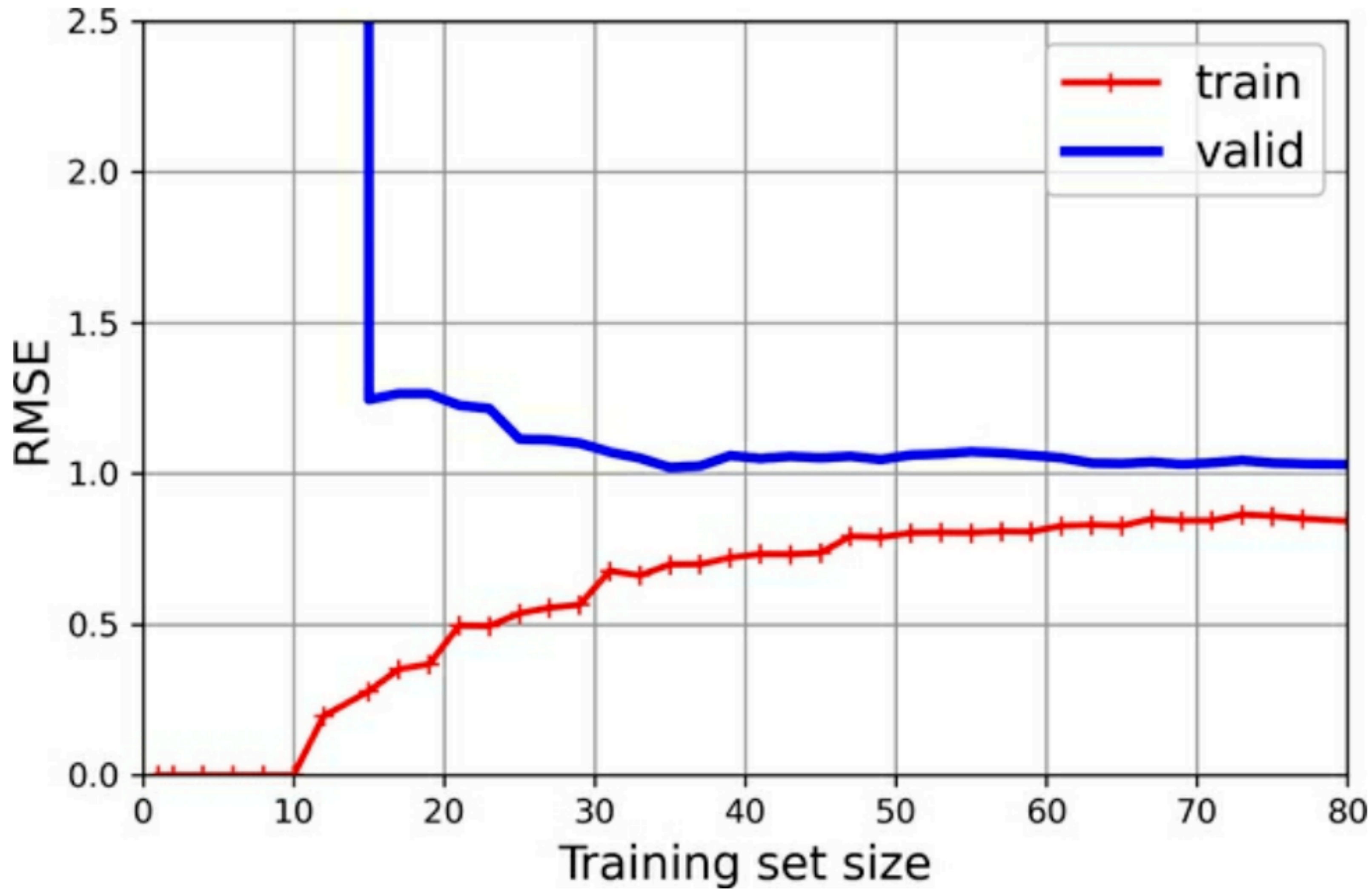
Underfitting: Linear Model, Quadratic Data

- Model can fit one or two training instances perfectly
- Model plateaus and stops improving when more data is added



Overfitting: 10th Degree Model, Quadratic Data

- Training error is always less than validation error
- Adding a lot more data can correct the overfitting



Types of Error

- ***Bias***
  - Caused by wrong assumptions, underfitting
  - Such as using a linear model to fit quadratic data
- ***Variance***
  - Model with too many parameters
  - Overfitting the training data
- ***Irreducible error***
  - Noise in the data
- Increasing a model's complexity will increase variance and reduce bias

Regularized Linear Models

Regularizing the Model

- Reduce the number of polynomial degrees
- For a linear model, constrain the weights of the model
- Three ways
  - ***Ridge regression***
  - ***Lasoo regression***
  - ***Elastic net regression***

Ridge Regression

Equation 4-8. Ridge regression cost function

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \frac{\alpha}{m} \sum_{i=1}^{n} {\theta_i}^2$$

- Squares of parameters are added to the cost function
- Model will keep the weights as small as possible
- Hyperparameter **α** controls amount of regularization
    - **α** of zero is just linear regression
    - Large **α** makes all weights small
- This is called the ℓ_2 norm, using the square of the weights

Various Levels of Ridge Regularization

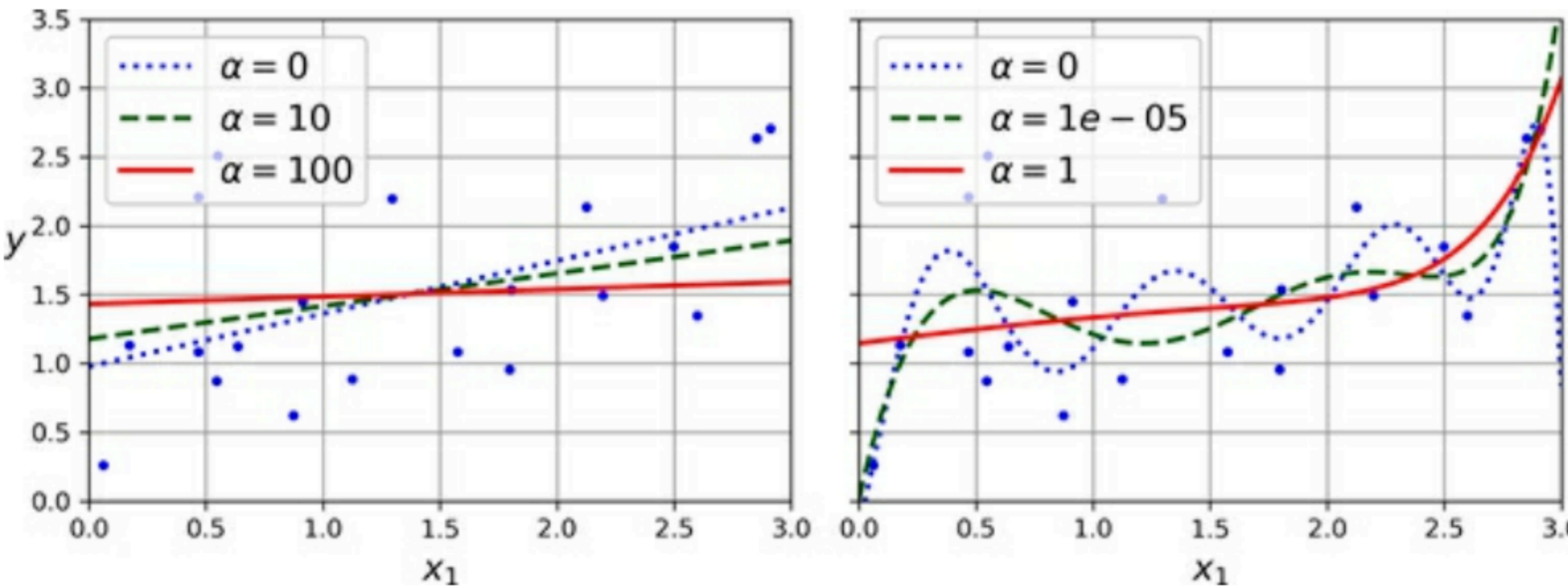


Figure 4-17. Linear (left) and a polynomial (right) models, both with various levels of ridge regularization

Lasso Regression

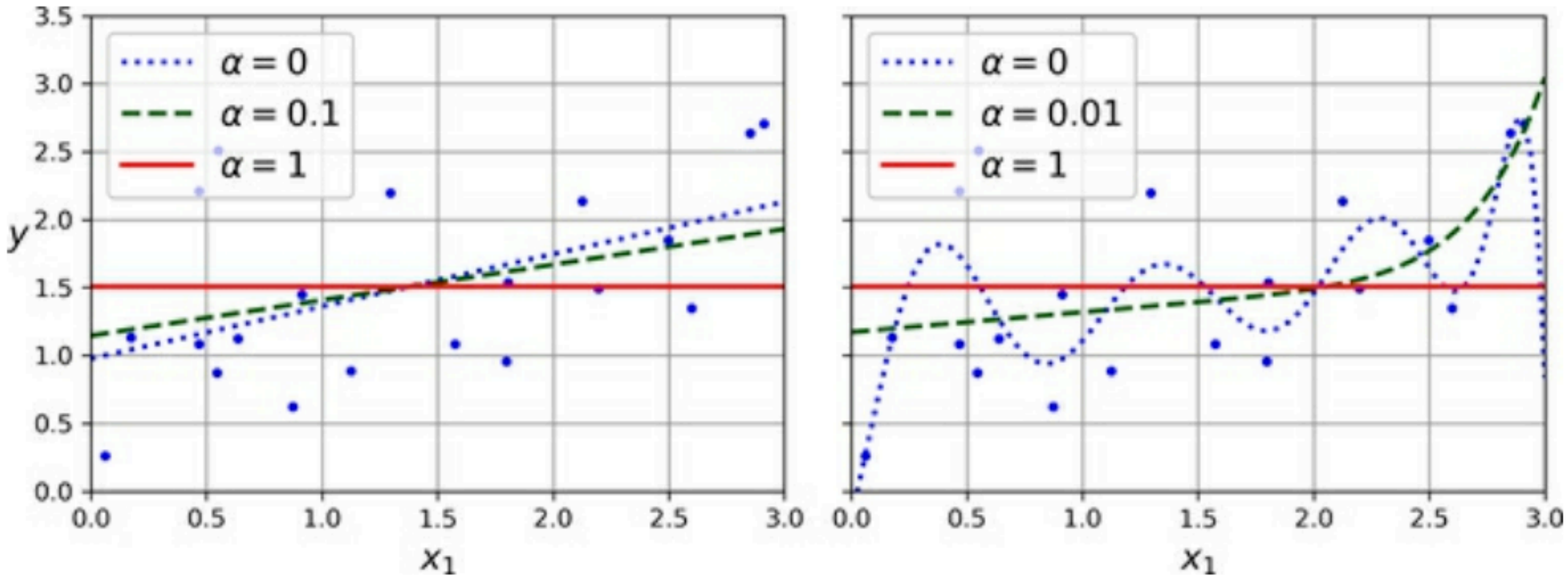
Equation 4-10. Lasso regression cost function

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + 2\alpha \sum_{i=1}^{n} |\theta_i|$$

- Least absolute shrinkage and selection operator regression
- Uses the ℓ_1 norm, using the absolute value of the weights

Lasso Regression

- Tends to eliminate the weights of the least important features



Elastic Net Regression

Equation 4-12. Elastic net cost function

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + r\left(2\alpha \sum_{i=1}^{n} |\theta_i|\right) + \left(1 - r\right)\left(\frac{\alpha}{m} \sum_{i=1}^{n} \theta_i^2\right)$$

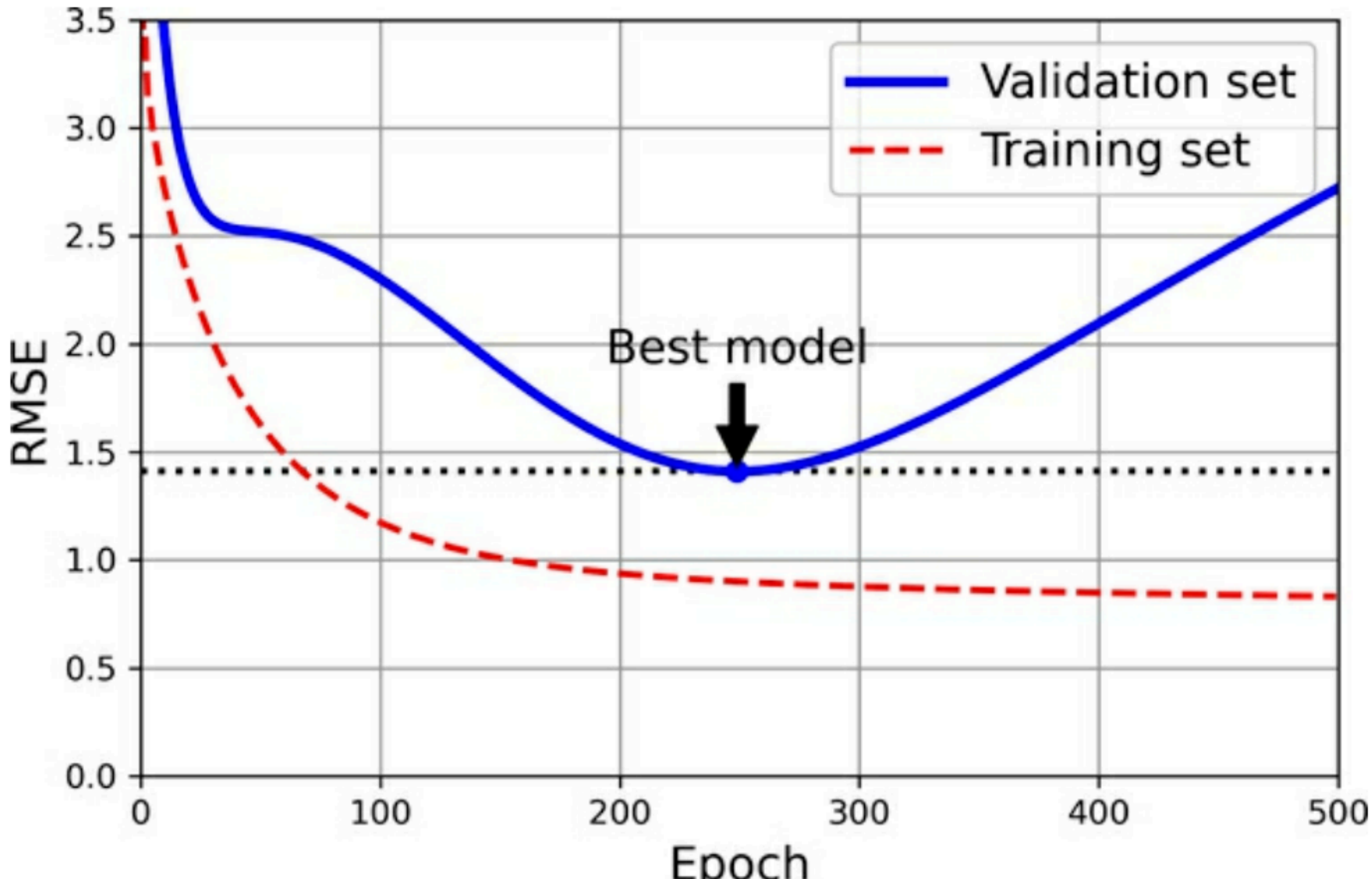
- Middle ground between ridge regression and lasoo regression
- Uses both ℓ_1 and ℓ_2 norms

Which to Use?

- Linear regression without regularization
  - Usually a bad choice
- Ridge regression is a good default
- If you suspect that only a few features are useful,
  - Use lasoo or elastic net
- Elastic net is preferred over lasoo
  - Lasoo can behave erratically
    - When the number of features is larger than the number of training instances, or
    - When several features are strongly correlated

Early Stopping

- A way to regularize gradient descent
- Stop training as soon as the validation error reaches a minimum



Logistic Regression

Classification

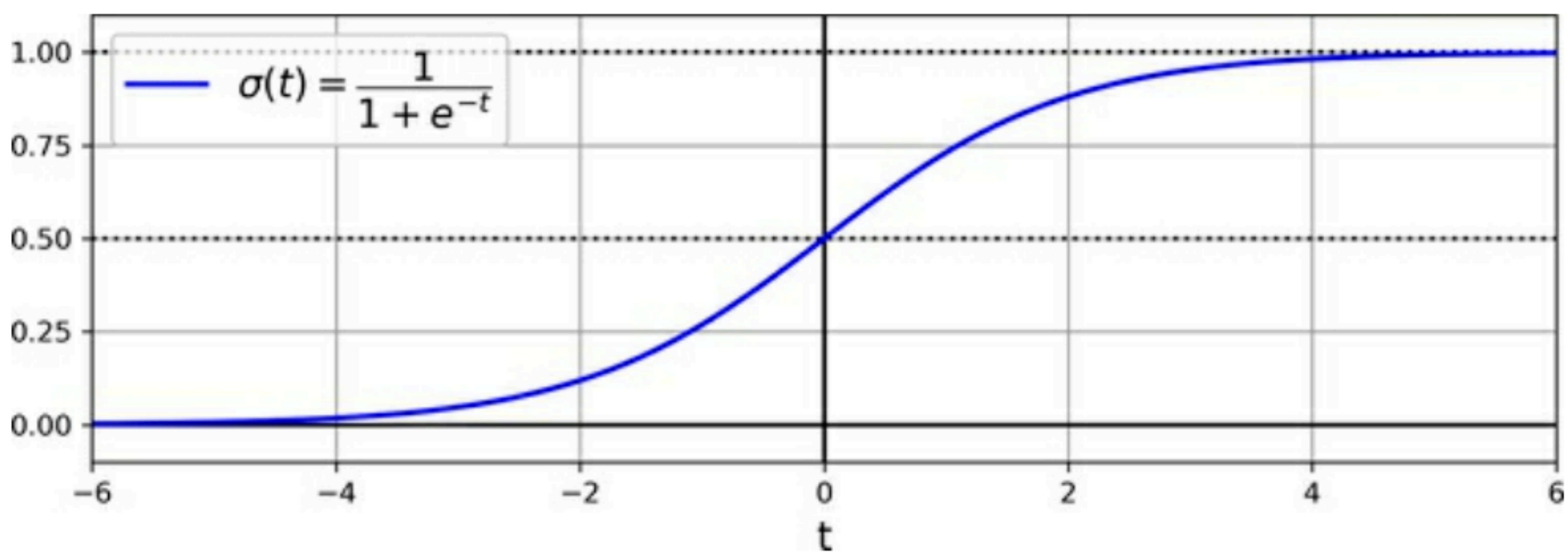
- A way to use a regression algorithm for classification
- Output of the regression measures probability of a classification
    - Such as whether an email is spam
- If the output is greater than a threshold (typically 50%)
    - The classification is ***positive class*** (True)

Logistic Function

Logistic regression model estimated probability (vectorized form)

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}\right)$$

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Cost Function

Equation 4-16. Cost function of a single training instance

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1-\hat{p}) & \text{if } y = 0 \end{cases}$$

- If model has a low probability for a positive instance
  - p is near zero, top row
  - Large cost
- If model has high probability for a negative instance
  - p is near one, bottom row
  - Large cost

Logistic Cost Function

- No closed-form solution
- But cost function is convex
- Gradient descent works well

Example: Iris Dataset

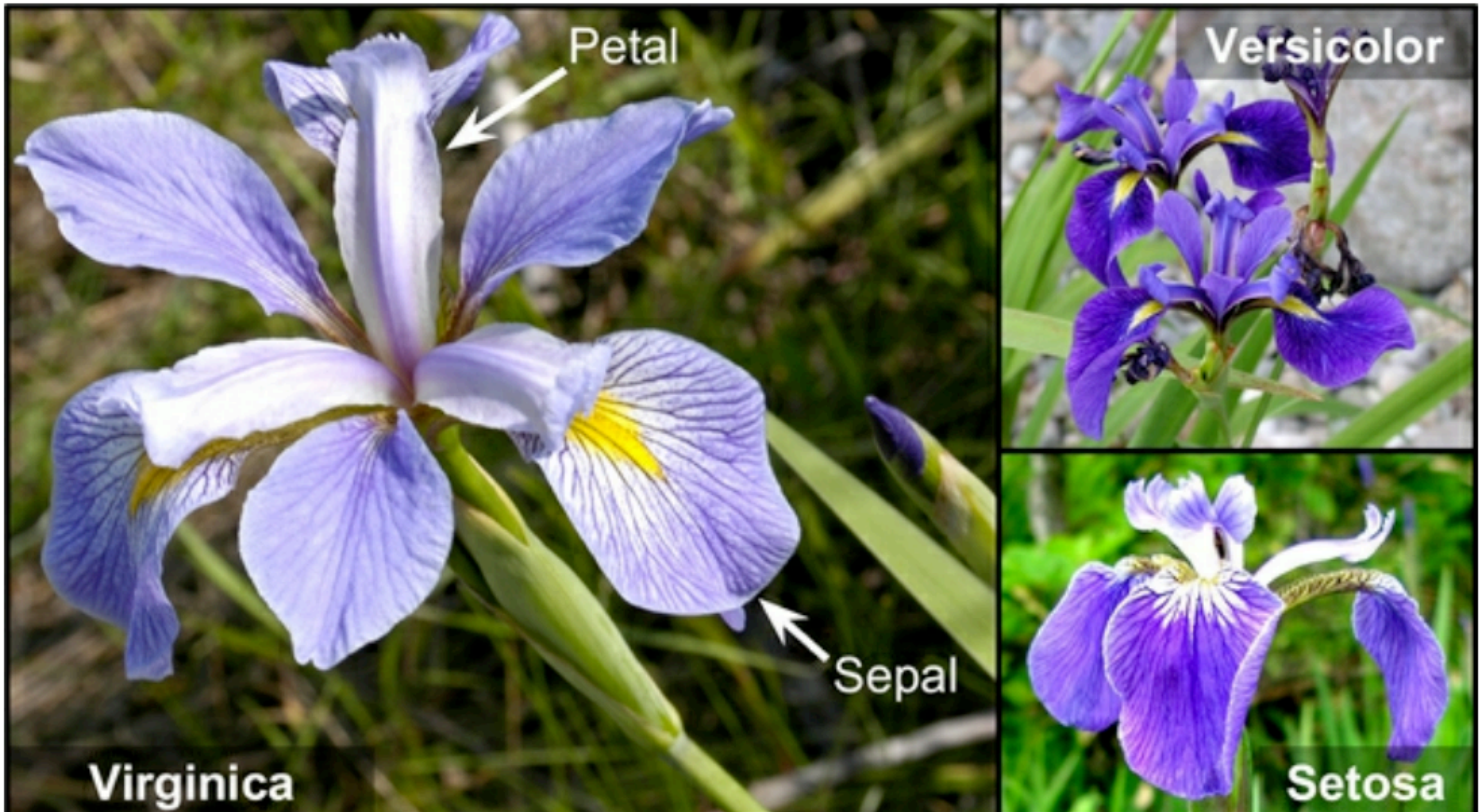


Figure 4-22. Flowers of three iris plant species[12]

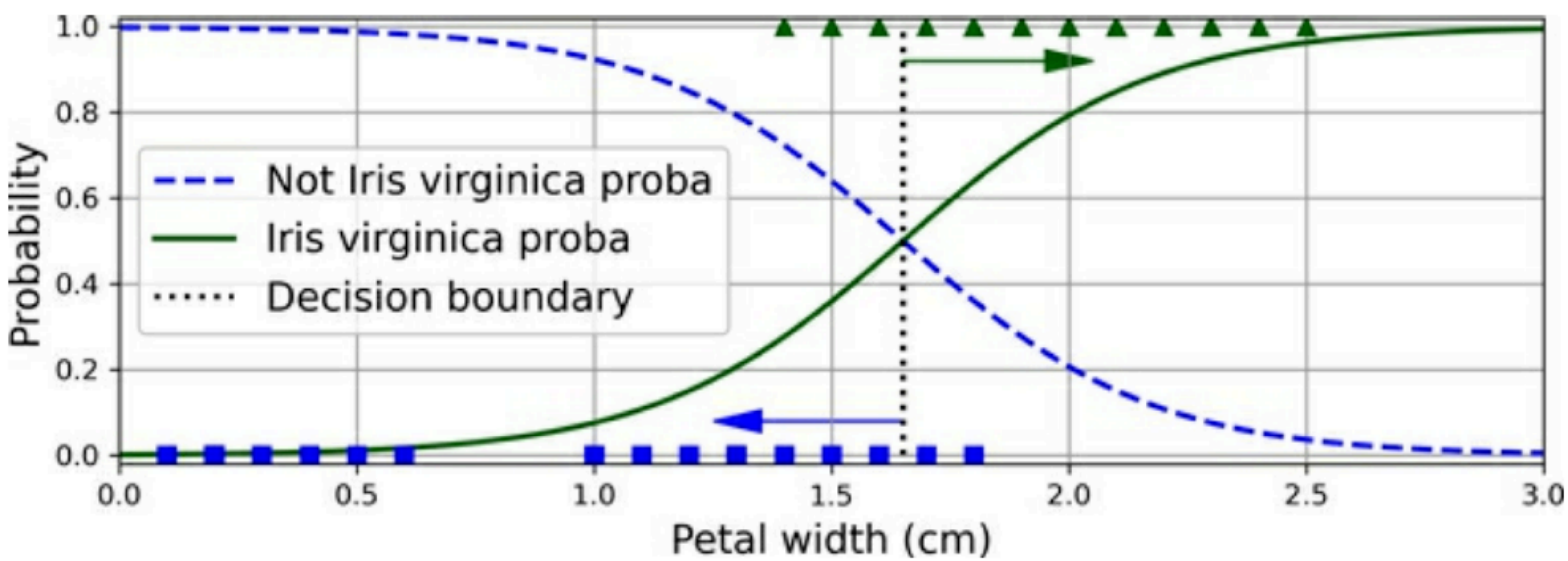
Three Features

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris(as_frame=True)
>>> list(iris)
['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
 'filename', 'data_module']
>>> iris.data.head(3)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
```

- Sepal width
- Petal length
- Petal width

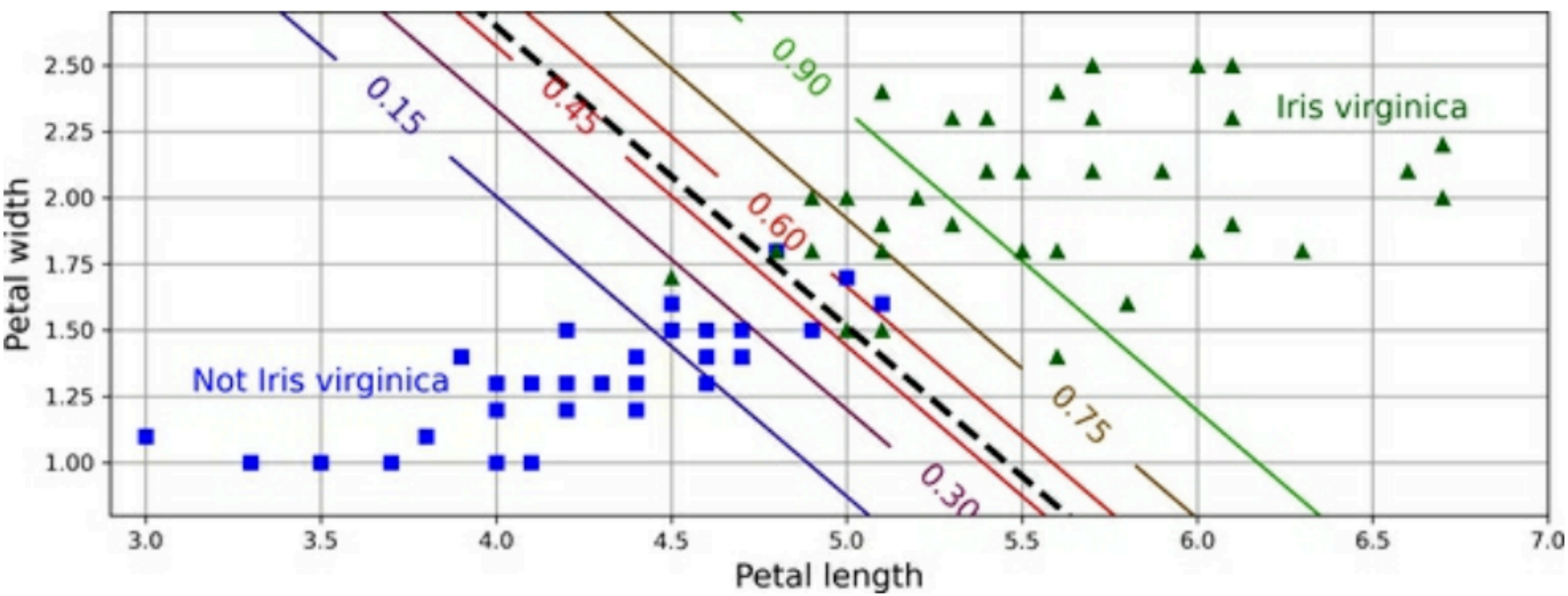
Using Only Petal Width

- *Iris virginica proba* has wider petals
- But there's considerable overlap



Using Petal Length and Petal Width

- Dashed line is 50% probability



Kahoot!

Ch 4b