Artificial Intelligence and Machine Learning for Networks and IoT

Class 2 — Linear Regression

Pedro Braconnot Velloso

Black box

"Statistical learning should not be viewed as a series of black boxes. No single approach will perform well in all possible applications. Without understanding all of the cogs inside the box, or the interaction between those cogs, it is impossible to select the best box. Hence, we have attempted to carefully describe the model, intuition, assumptions, and trade-offs behind each of the methods that we consider."

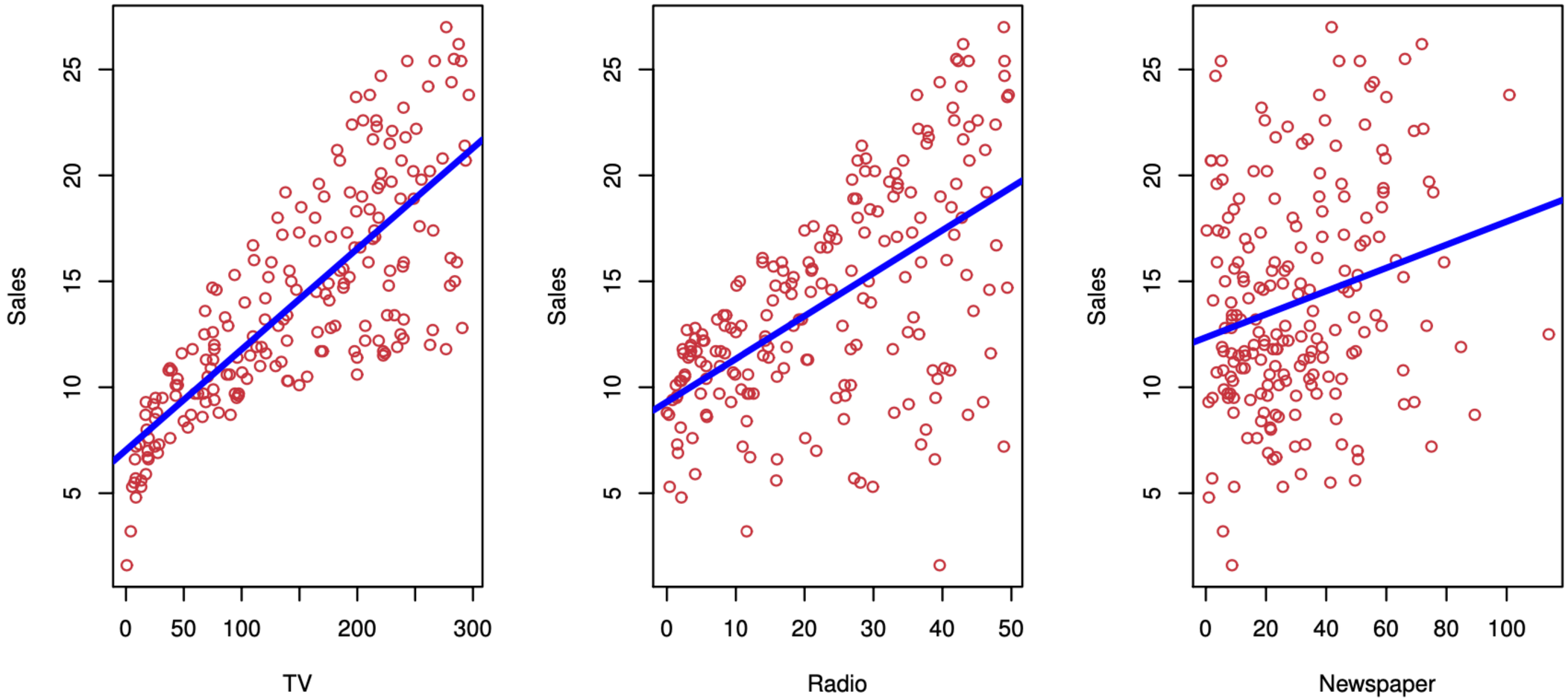
Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani,
"An Introduction to Statistical Learning: with Applications in R", Springer, 2nd edition

Linear Regression

Linear regression

- A very simple approach for supervised learning
- It is a useful tool for predicting a quantitative response
- It serves as a good jumping-off point for newer approaches
  - Many fancy statistical learning approaches can be seen as
    - Generalizations or extensions of linear regression

Advertising data set



source: James, G., Witten, D., Hastie, T., Tibshirani, R. "An Introduction to Statistical Learning "

Some important questions

- Is there a relationship between advertising budget and sales?
- How strong is the relationship between advertising budget and sales?
- Which media are associated with sales?
- How large is the association between each medium and sales?
- How accurately can we predict future sales?
- Is the relationship linear?
- Is there synergy among the advertising media?

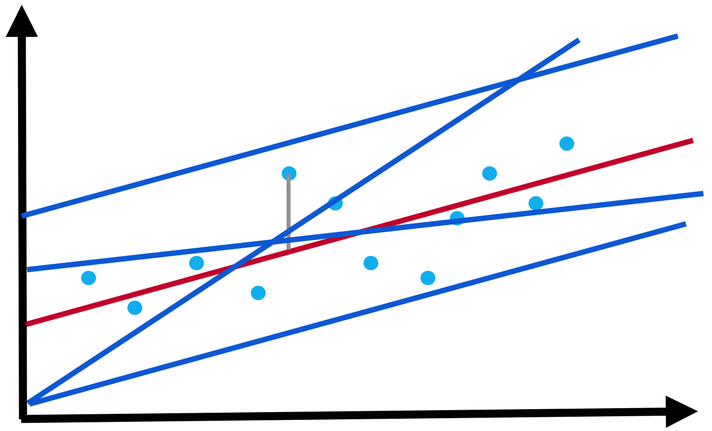
Simple Linear Regression

- An approach for predicting a quantitative response **Y**
  - On the basis of a single predictor variable **X**
- It assumes an approximately linear relationship between **X** and **Y**

$$Y \approx \beta_0 + \beta_1 X$$

- $\beta_0 \; and \; \beta_1$ —> unknown parameters

$$sales \approx \beta_0 + \beta_1 TV$$



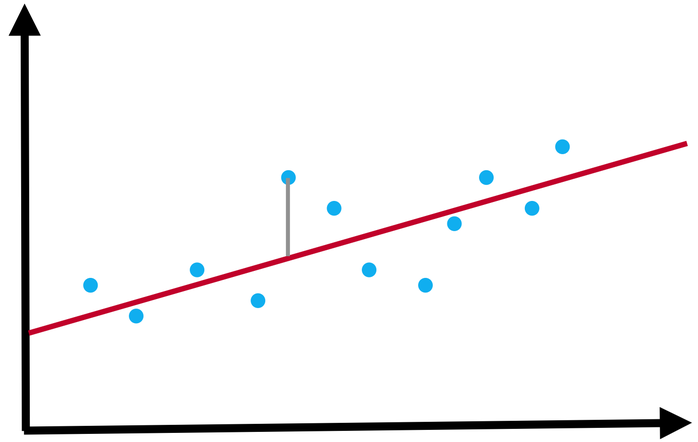
Estimating the model parameters

- We use our training data to produce estimates for
  - The model parameters

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

  - Remember
    - We have a set of **n** pairs

$$(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$$

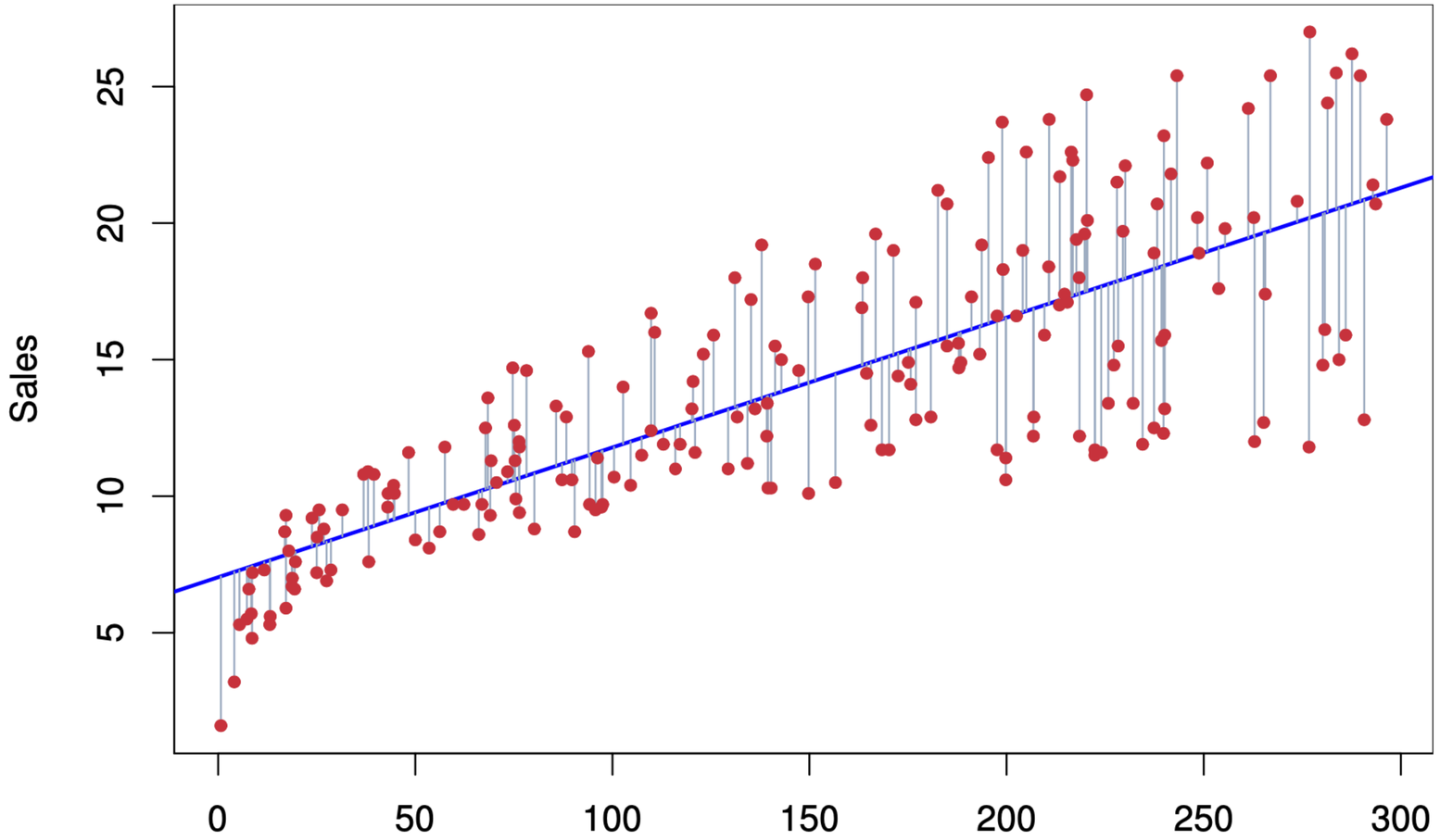


Measuring closeness

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

i-th Residual

$$e_i = y_i - \hat{y}_i$$



source: James, G., Witten, D., Hastie, T., Tibshirani, R. "An Introduction to Statistical Learning "

Residual Sum of Squares — RSS

- The sum of residual errors

$$RSS = \sum_{i=1}^{n} {e_i}^2$$

- Rewriting the previous equation
  - Cost function
    - Want to minimize

$$J(\beta) = \frac{1}{2} \sum_{i=1}^{n} (h_\beta(x^{(i)}) - y^{(i)})^2$$

Linear regression

- General form

Weights

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_d x_d$$

Hypothesis

Intercept term

Vector form

$$h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T x$$

$$x_0 = 1$$

- With two features

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

How do we learn the values of θ

- Measure how close the model gets — > cost function
  - RMSE — Root Mean Square Error
  - Residual Sum of Squares
  - Least Squares cost
- Let's take a look at the Mean Square Error

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

Normal equation

- Closed-form solution

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

- Computing the inverse of a matrix
  - The matrix $X^\intercal X$ must be invertible
    - Instances (m) >= features (n)
      - Some features are redundant
  - High computational complexity ($O(n^{2.4})$ to $O(n^3)$)
    - Multiply by 5.3 to 8 computation time when we double the **num_features**
- Not appropriate for large datasets and large number of features

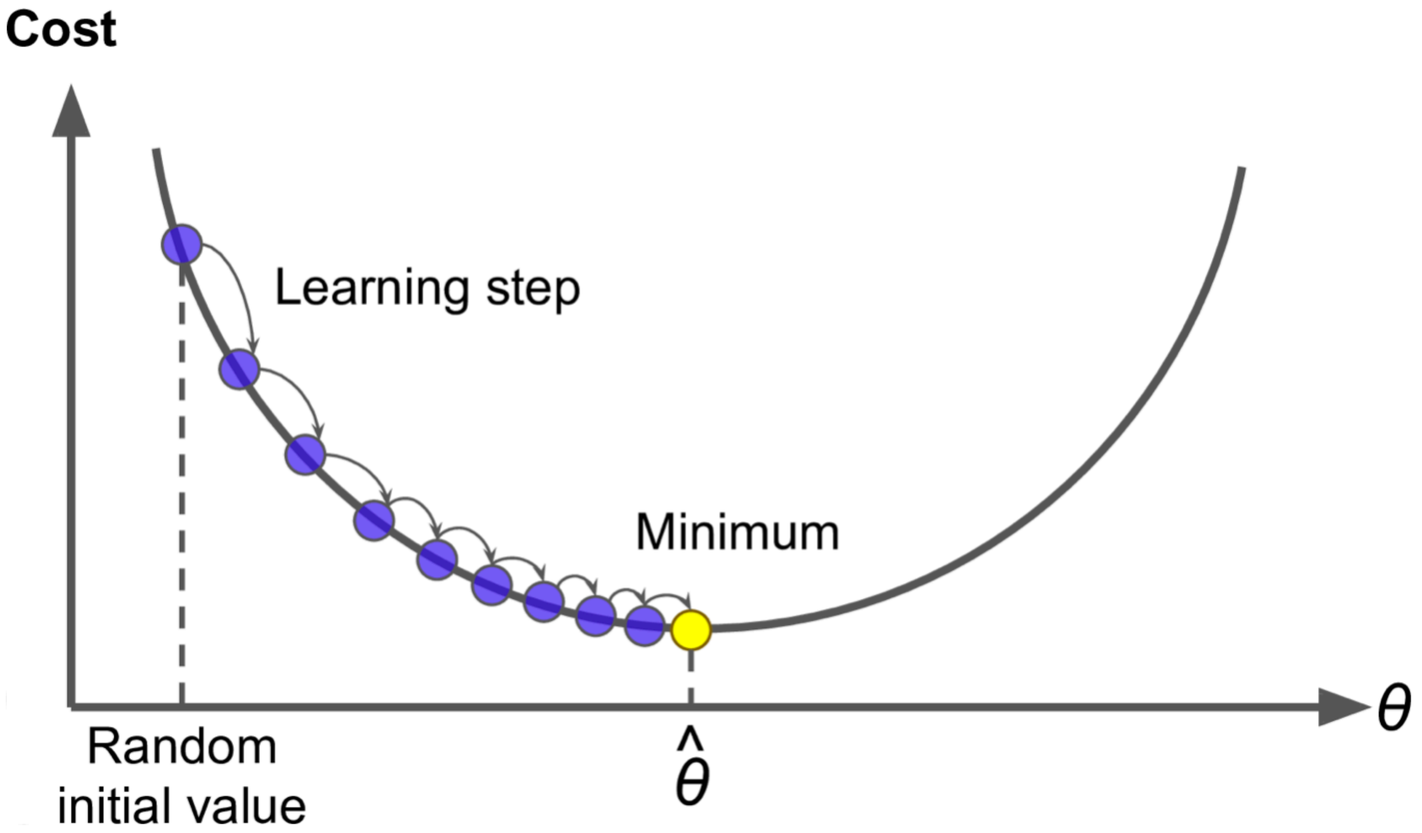
Linear regression in Scikit-learn

- Implements a pseudo inverse
  - Singular Value Decomposition (SVD)
    - A standard matrix factorization technique
  - More efficient ($O(n^2)$)
  - Handles edge cases nicely

Gradient Descent

- Gradient Descent is a generic optimization algorithm
  - Capable of finding optimal solutions to a wide range of problems
- Interactive algorithm
  - Change function parameters
  - Minimize a cost function
- Basic idea
  - Suppose we are lost in the mountains in a dense fog
  - We can only feel the slope of the ground below your feet
    - Go downhill in the direction of the steepest slope

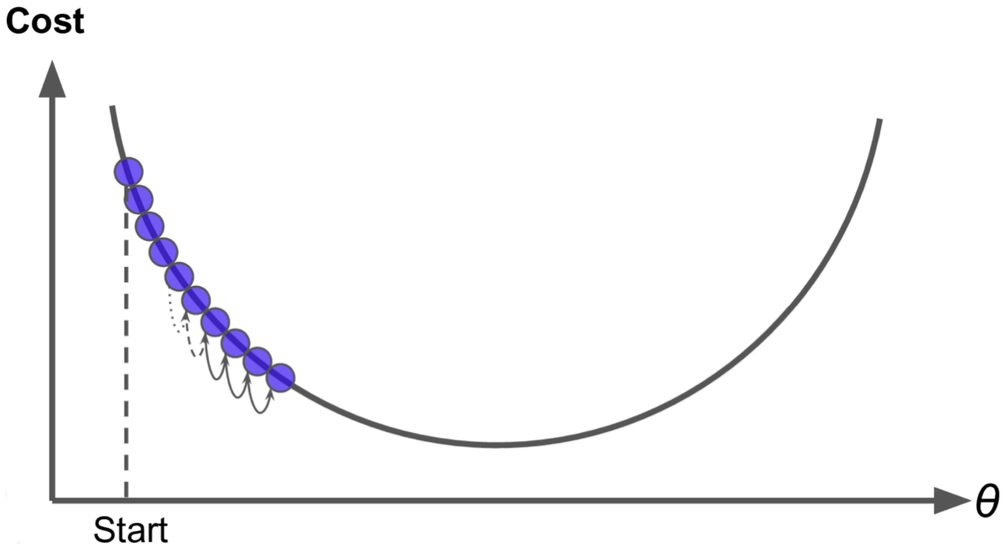
Gradient Descent



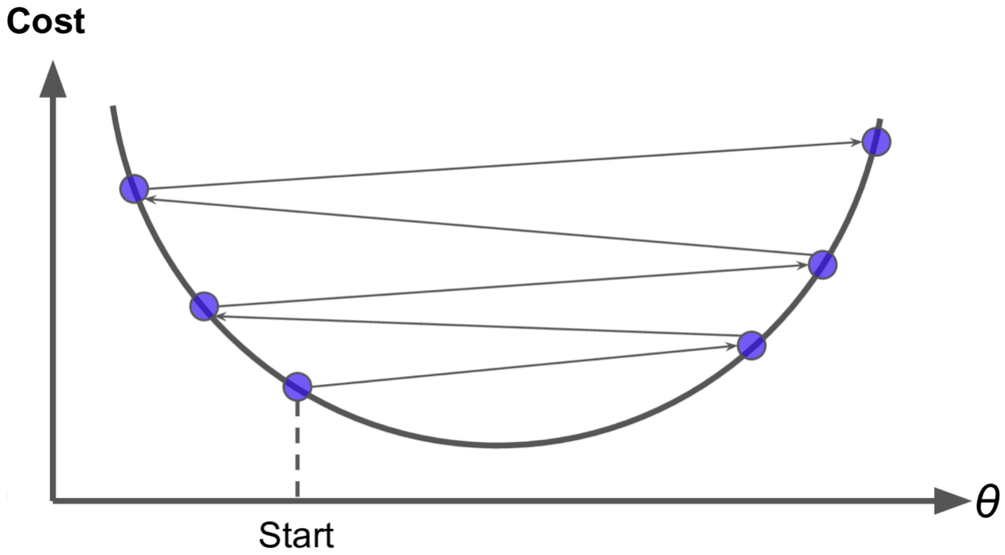
source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Learning rate

Too small

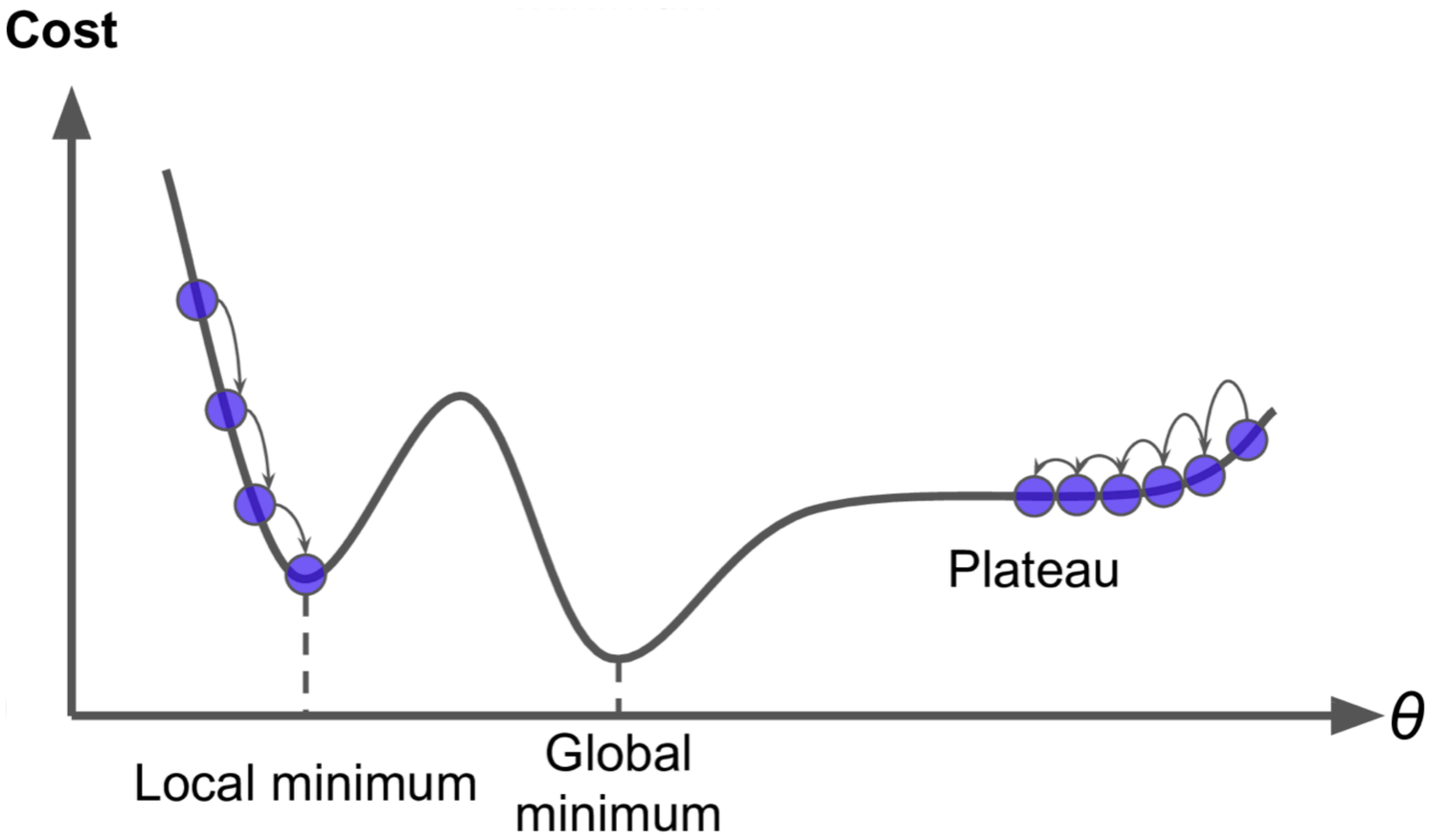


Too high



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Gradient Descent pitfalls



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Gradient Descent

- A cost function

$$J(\theta_0, \theta_1)$$

- Minimize

$$min\ J(\theta_0, \theta_1)$$

- Start with some value for θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$
  - Until we hopefully end up at a minimum

Gradient Descent

- Going in the direction of the negative gradient of the cost function

$$-\nabla J(\theta_0, \theta_1)$$

- The algorithm

Learning rate

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

- As we approach a local minimum
  - GD automatically take smaller steps
  - No need to decrease the learning rate over time

Gradient Descent

- Linear regression model
  - Cost function (MSE)

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- Partial derivatives of the cost function

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{2}{m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})\, x_j^{(i)}$$

Batch Gradient Descent

$$\frac{2}{m} X^T (X\theta - y)$$

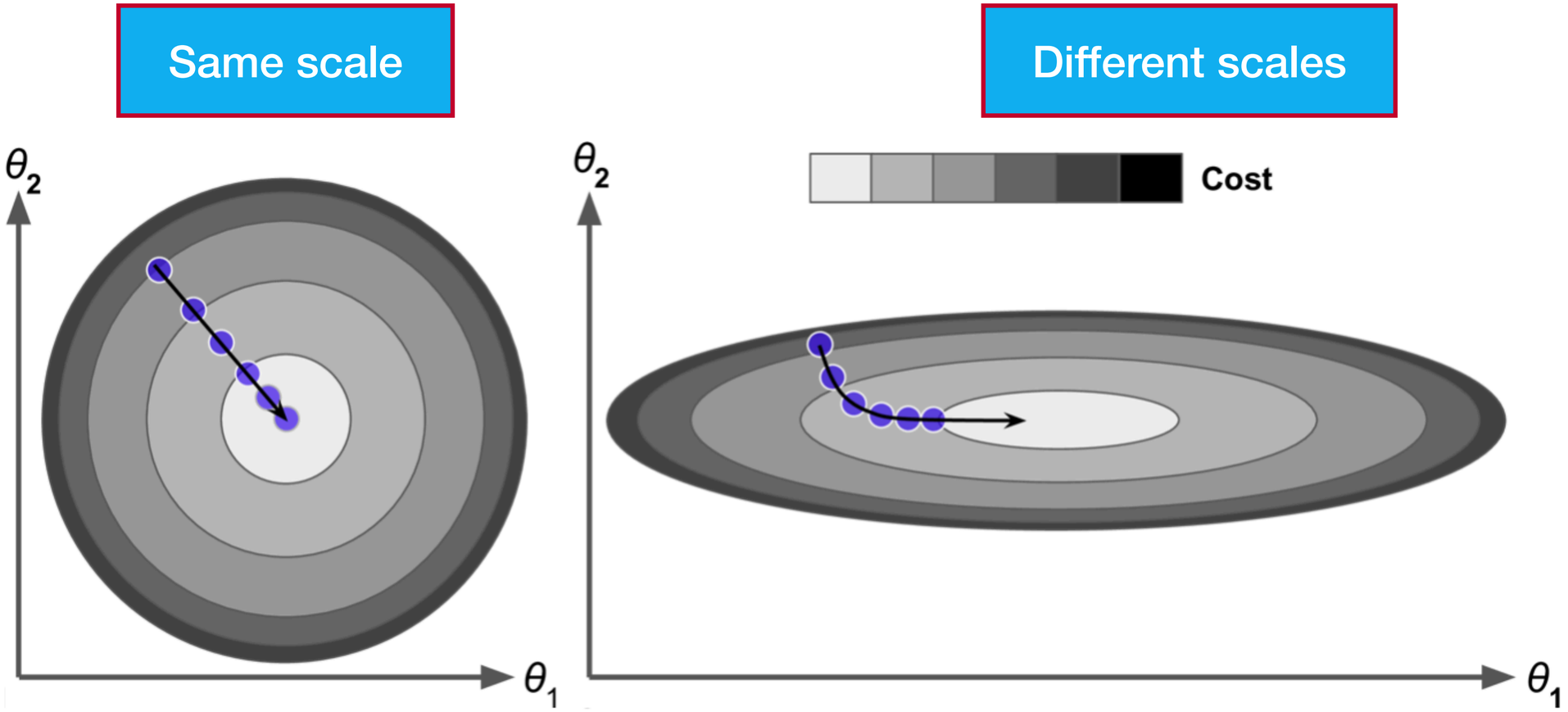
Gradient Descent

```
eta = 0.1  # learning rate
n_epochs = 1000
m = len(X_b)  # number of instances

np.random.seed(42)
theta = np.random.randn(2, 1)  # randomly initialized model parameters

for epoch in range(n_epochs):
    gradients = 2 / m * X_b.T @ (X_b @ theta - y)
    theta = theta - eta * gradients
```

Convergence

- Depends on several parameters
  - Learning rate
  - Number of parameters —> dimension
  - Difference in the scale of parameters

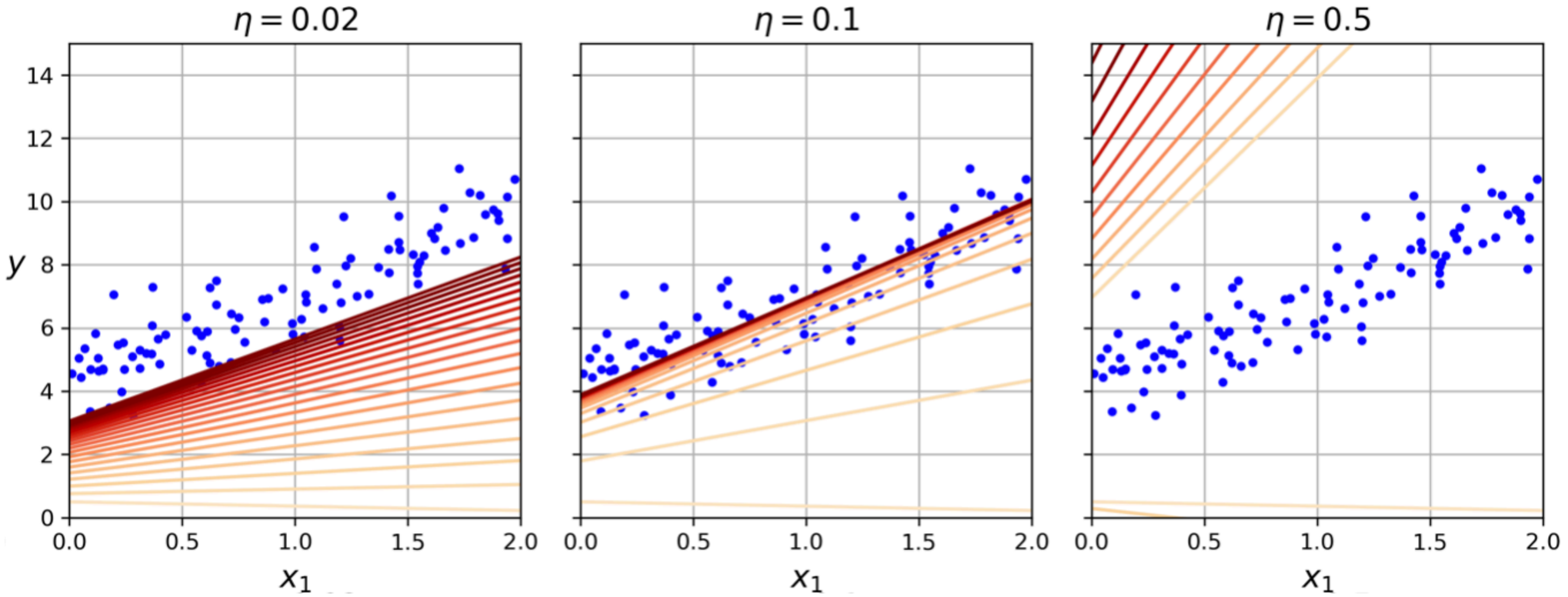
Cost function shape



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Gradient Descent with different LRs

- The first 20 steps with different learning rates



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Finding the learning rate

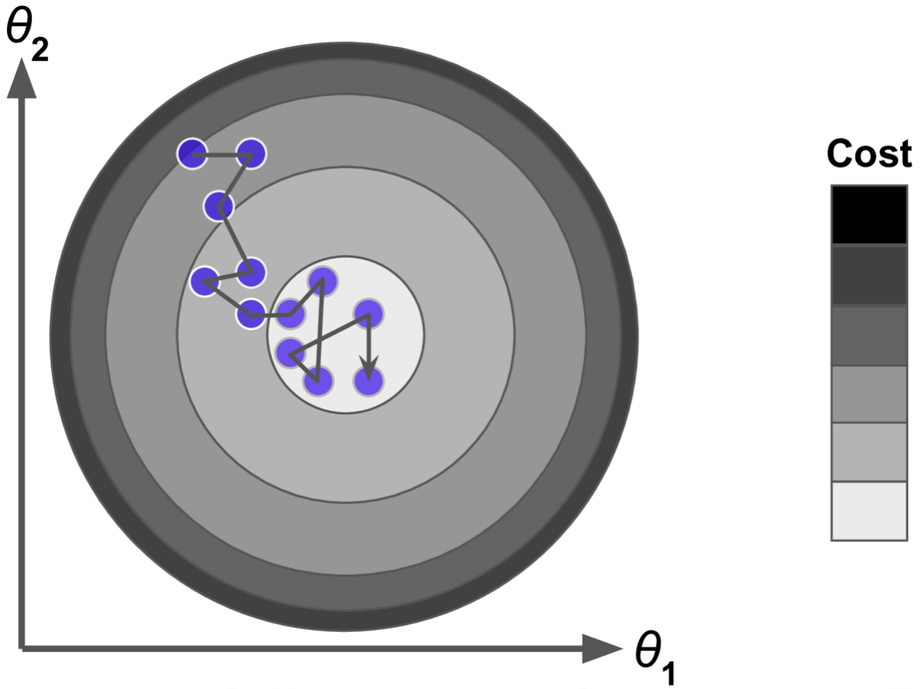
- We can use grid search
  - Limit the number of epochs
    - Grid search can eliminate models that take too long to converge
- A simple solution is to set a very large number of epochs
  - Interrupt the algorithm when the gradient vector becomes tiny
    - Smaller than a tiny number ε —> tolerance
    - It can take O(1/ε) iterations to reach the optimum
      - Within a range of ε
      - Depending on the shape of the cost function

Stochastic Gradient Descent

- The main problem with Batch Gradient Descent
  - It uses the whole training set
    - To compute the gradients at every step
      - Very slow when the training set is large
- SGD picks a random instance at each step
  - Much faster
  - Possible to train on huge datasets

Stochastic Gradient Descent

- The cost function will bounce up and down
  - Decreasing only on average



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Stochastic Gradient Descent

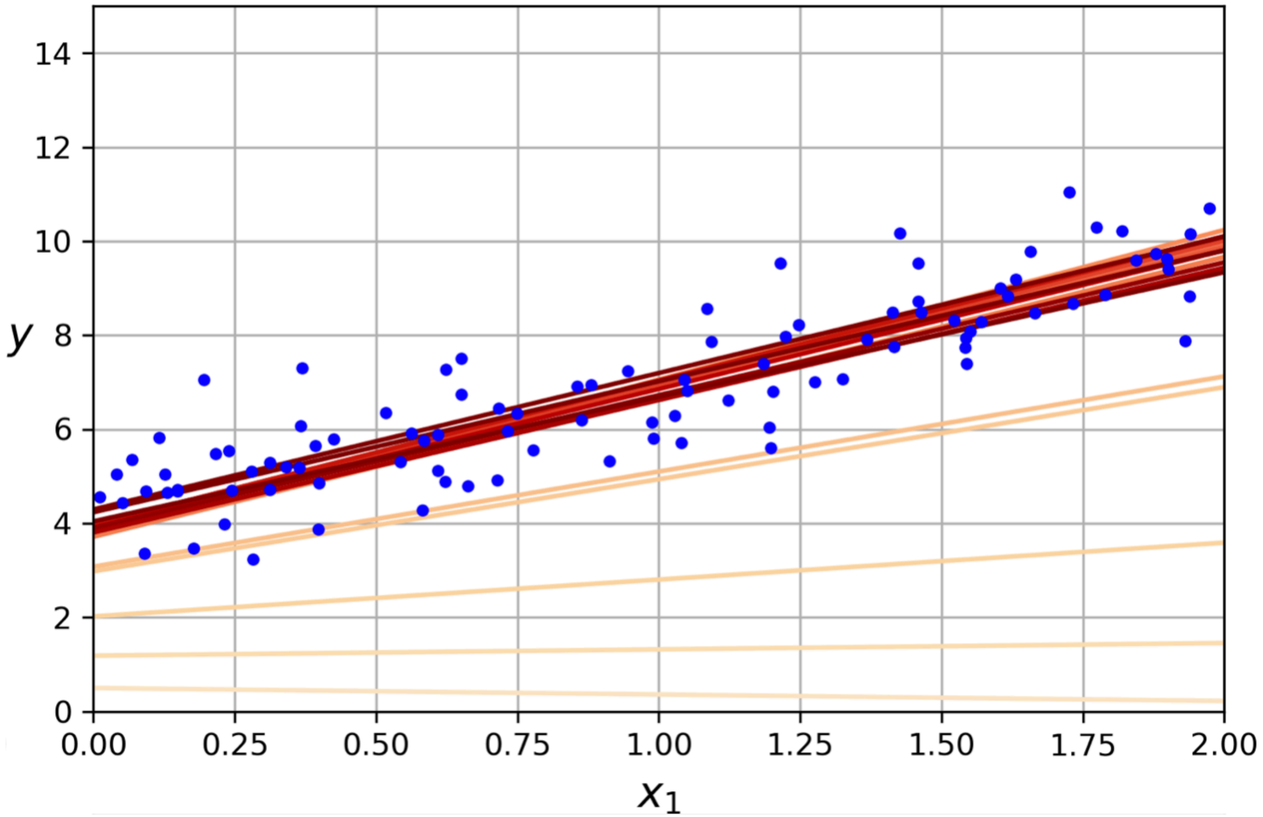
- It might be interesting for irregular cost functions
  - Jumping from local minima
- It might never settled at the minimum
- One solution to this dilemma
  - Gradually reduce the learning rate
    - The steps start large
      - Helps make quick progress and escape local minima
      - Get smaller and smaller
        - Allowing the algorithm to settle at the global minimum

Stochastic Gradient Descent

- Learning schedule
  - The function that determines the learning rate at each iteration
  - If the learning rate is reduced too quickly
    - We may get stuck in a local minimum
    - End up frozen halfway to the minimum
  - If the learning rate is reduced too slowly
    - We may jump around the minimum for a long time
      - End up with a suboptimal solution if you halt training too early

Stochastic Gradient Descent

- The first 20 iterations



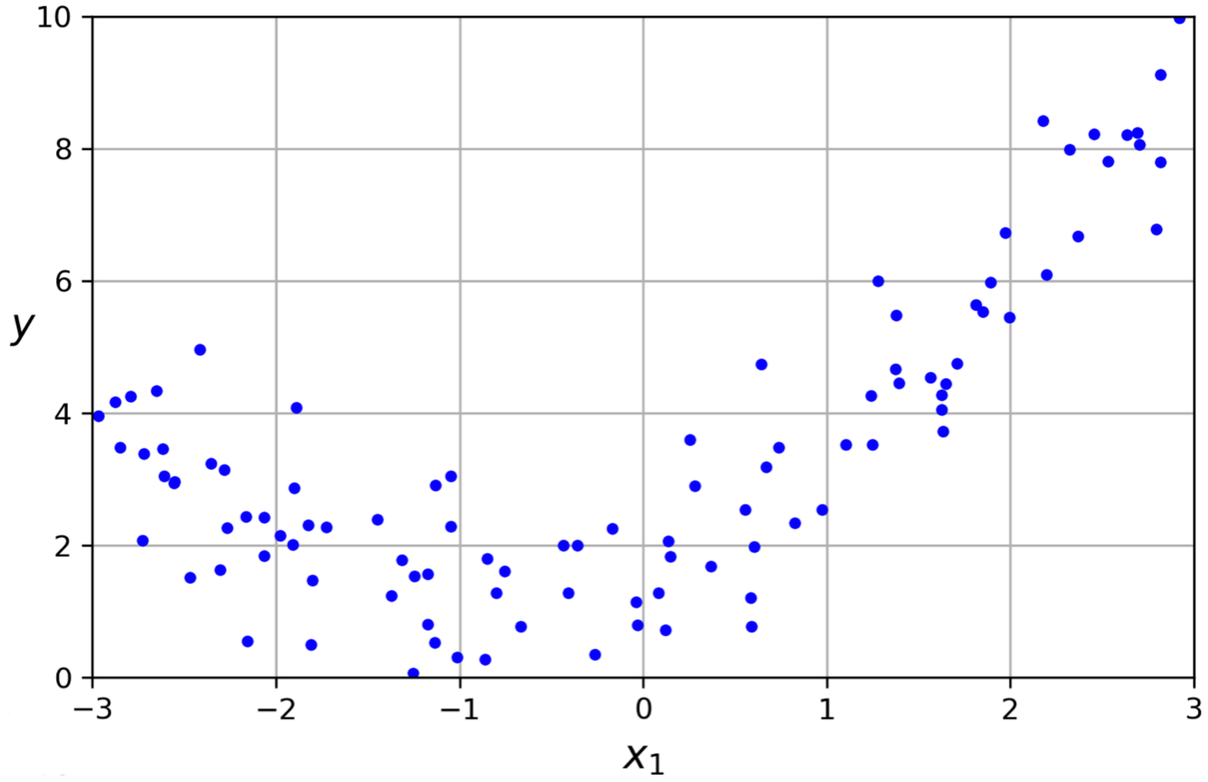
source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Polynomial Regression

- We can use a linear model to fit nonlinear data
- Train a linear model on an extended set of features
  - Add powers of each feature as new features

Polynomial Regression

$$y = 0.5{x_1}^2 + 1.0x_1 + 2.0 + \text{Gaussian noise.}$$

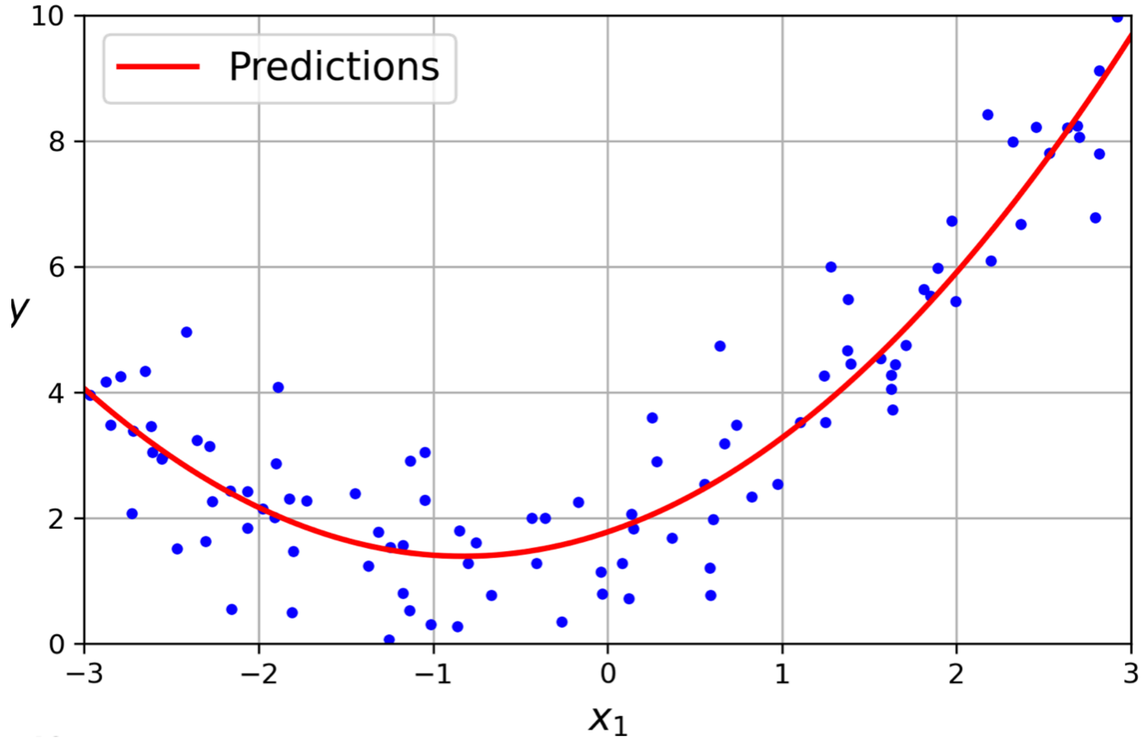


Polynomial Regression

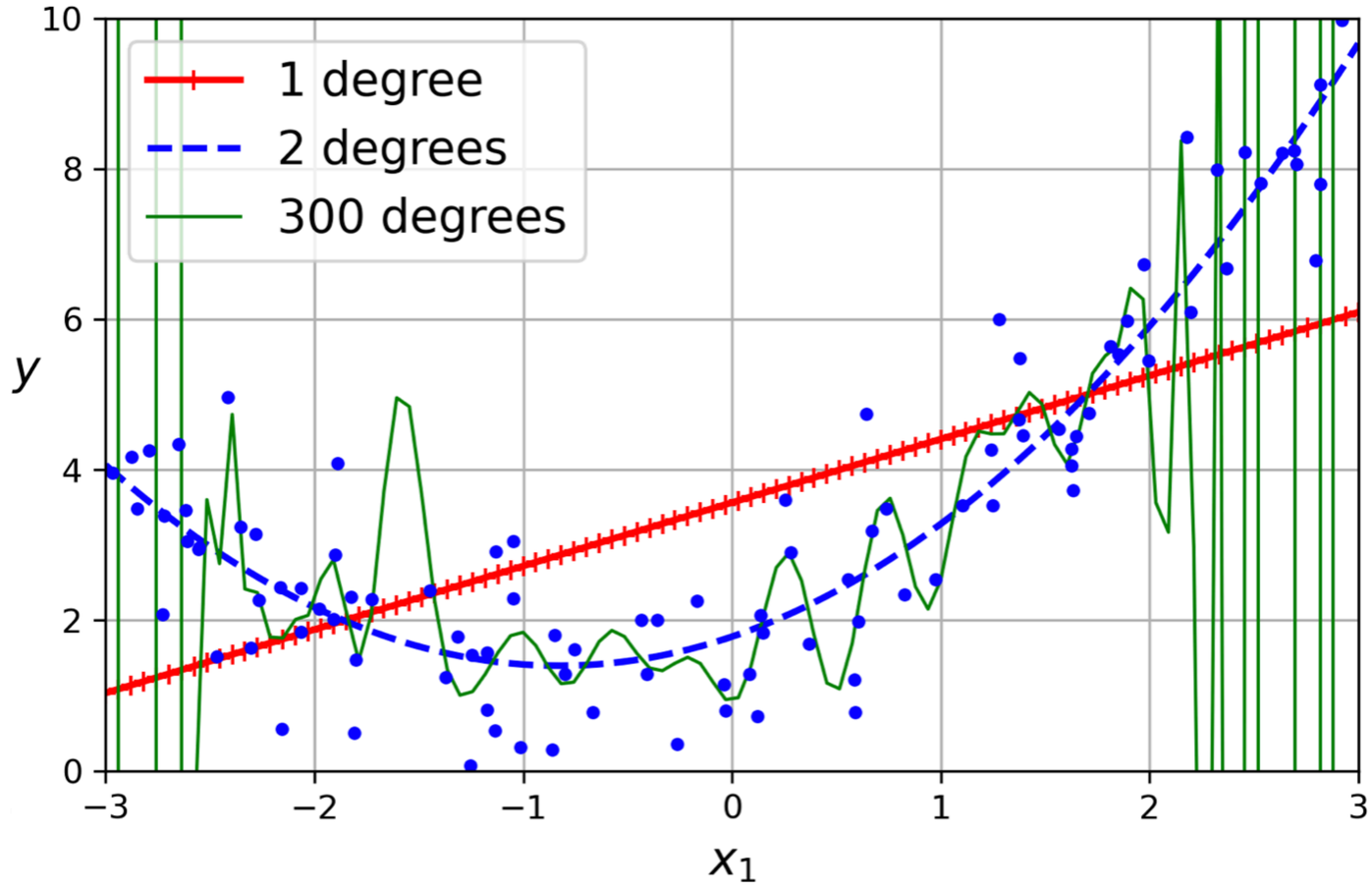
- Linear regression with two features: x and x^2

$$y = 0.5{x_1}^2 + 1.0x_1 + 2.0$$

$$\hat{y} = 0.56x^2 + 0.93x + 1.78$$



How to decide the complexity



How to decide the complexity

- We will use cross-validation to get an estimate
  - The model's generalization performance
- If a model performs well on the training data
  - But generalizes poorly according to the cross-validation metrics
    - The model is overfitting
- If it performs poorly on both
  - It is underfitting
- This is one way to tell when a model is too simple or too complex

How to decide the complexity

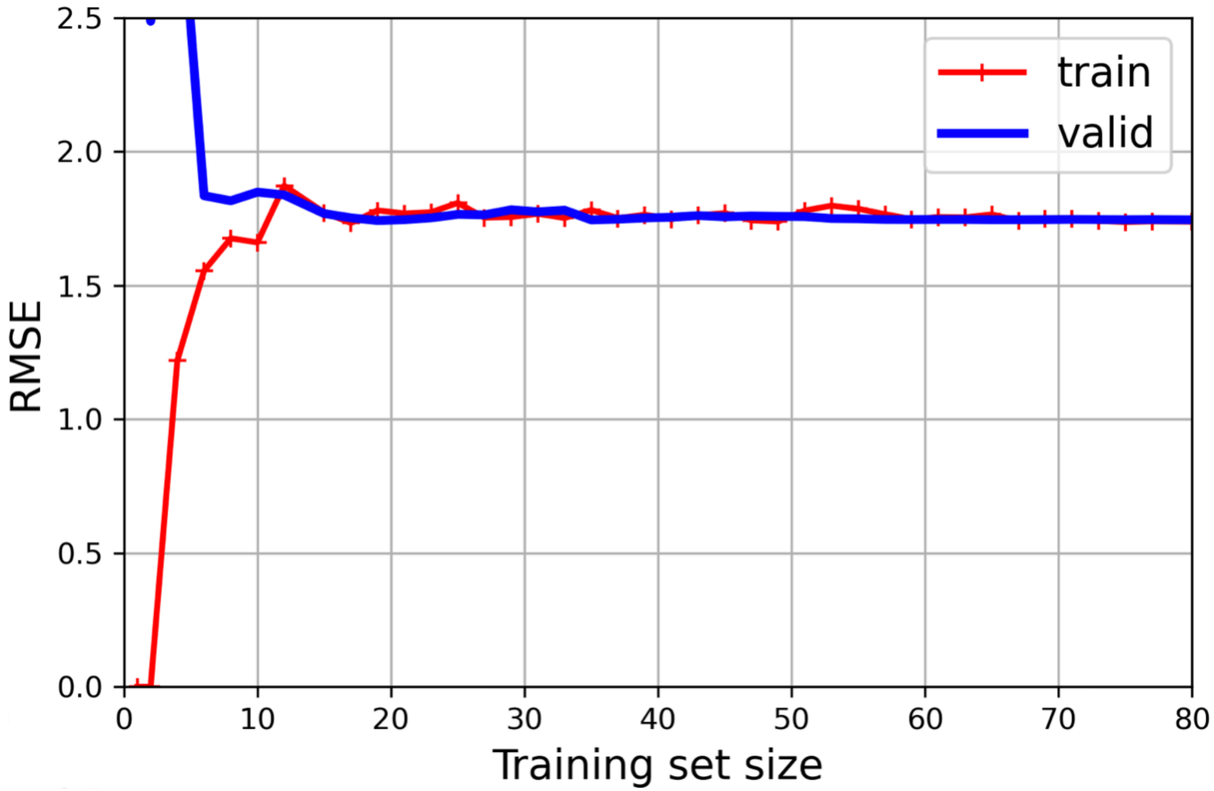
- Look at the **learning curves**
  - Plots of the model's training error and validation error
    - As a function of the training iteration
  - Evaluate the model at regular intervals during training
    - Both the training set and the validation set
    - Plot the results
- If the model cannot be trained incrementally
  - It does not support **partial_fit()** or warm_start
    - Train the model several times on gradually larger subsets of the training set

learning_curve()

- It trains and evaluates the model using cross-validation
- By default it retrains the model on growing subsets of the training set
  - If the model supports incremental learning we can set
    - **exploit_incremental_learning=True**
      - When calling **learning_curve()**
    - To train the model incrementally
- Returns the training set sizes
  - The training and validation scores for each size

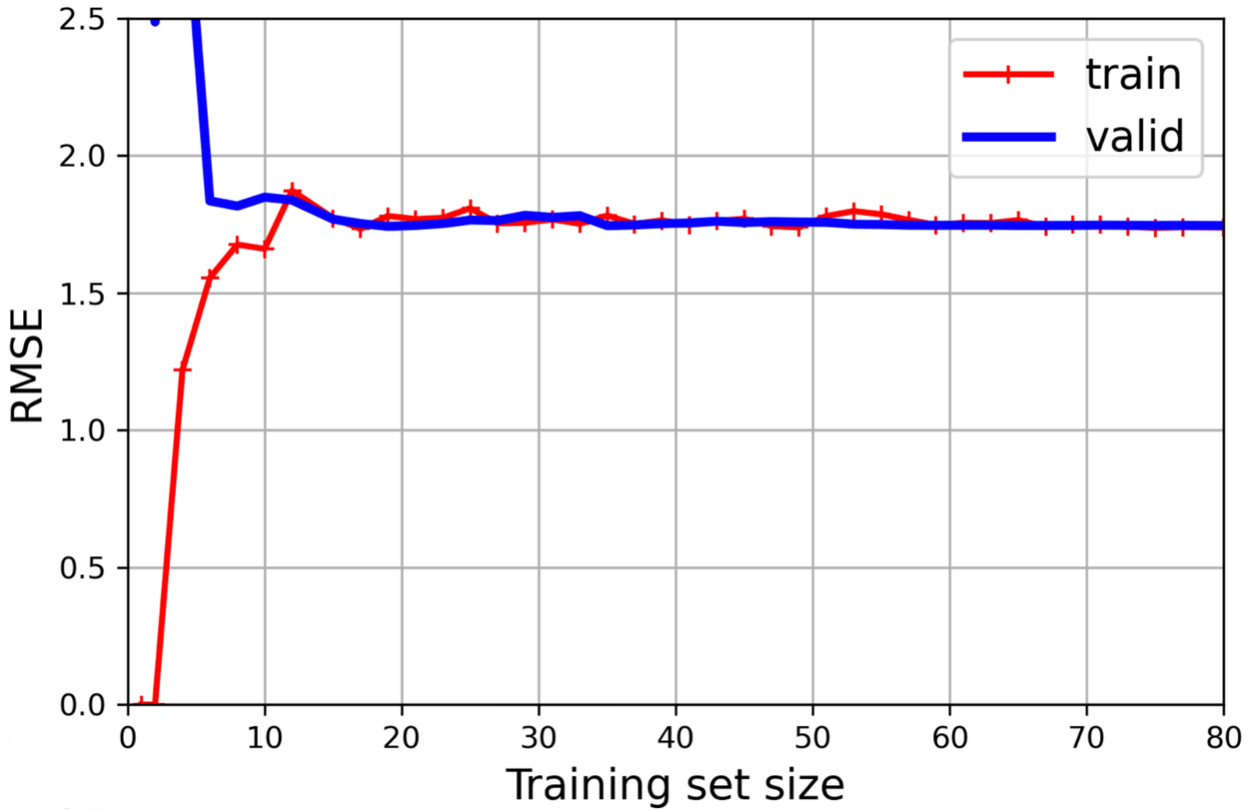
learning_curve()

- Linear model



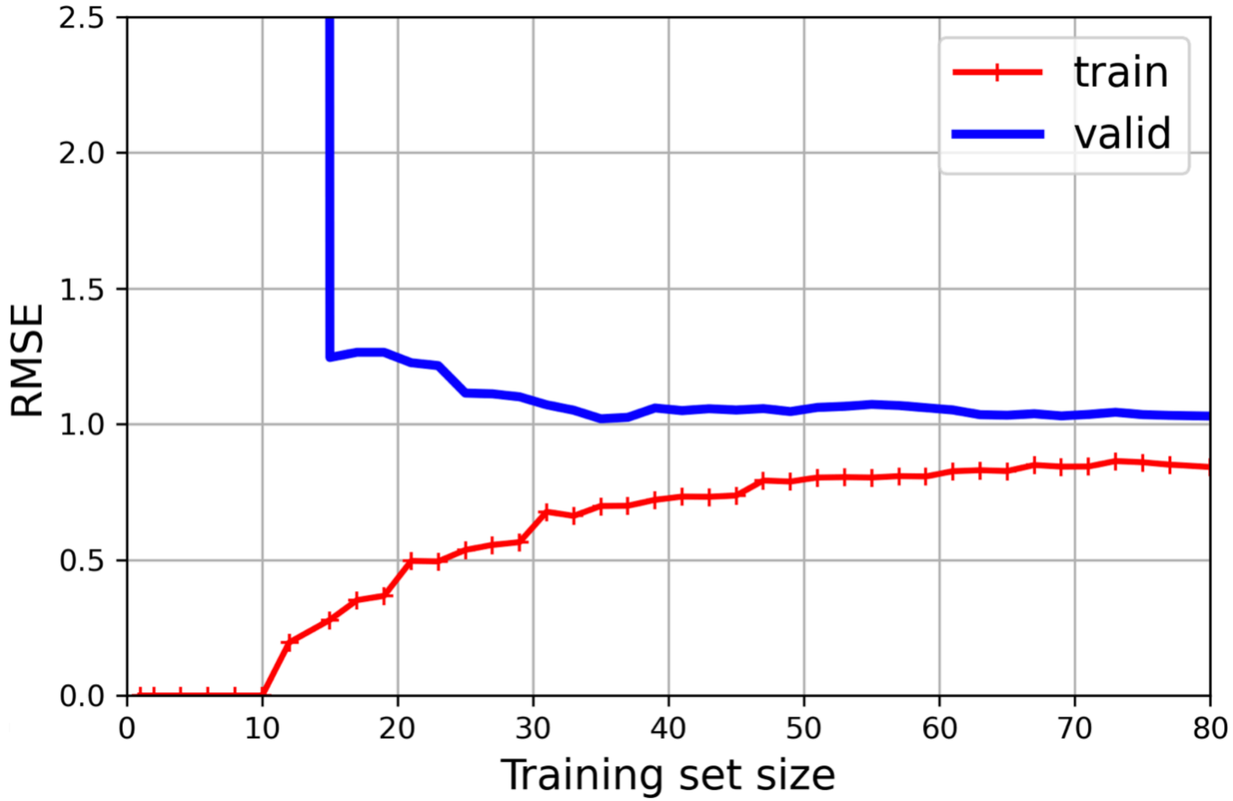
Typical of a model that's underfitting

- This model is underfitting
- The training error
- The validation error
- Both reach a plateau
  - High



learning_curve()

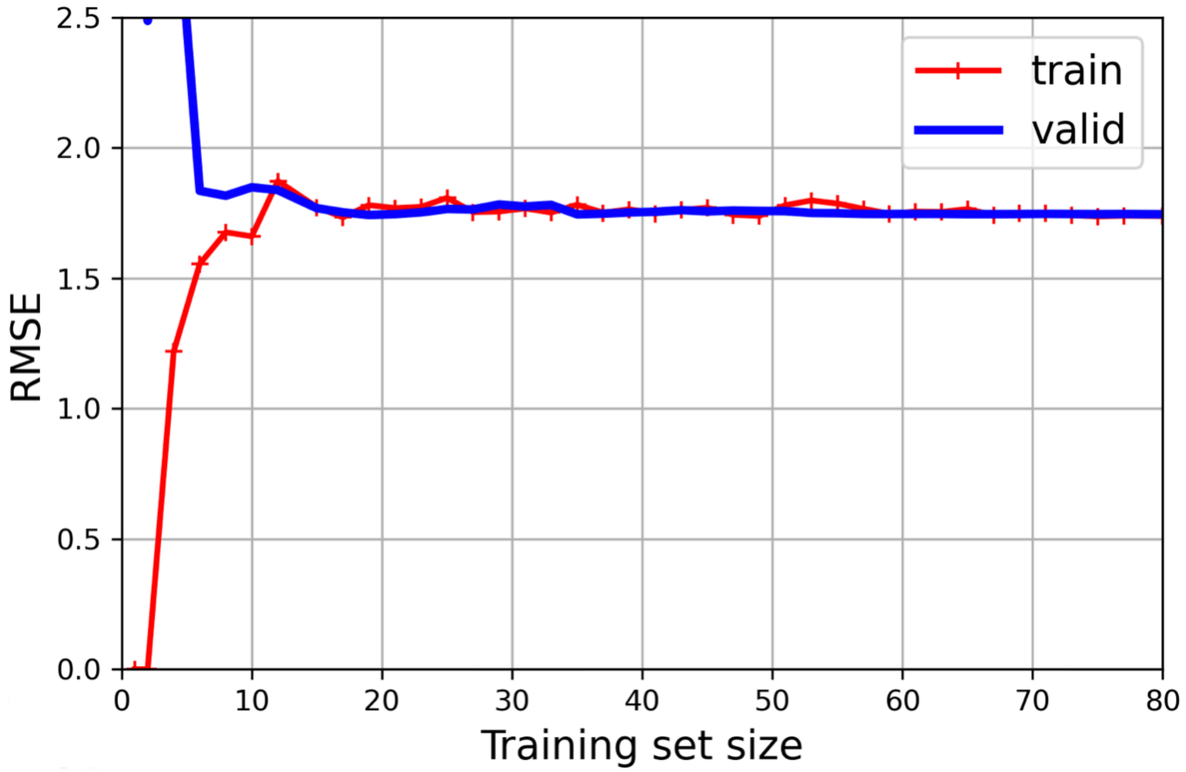
- Polynomial model —> degree = 10



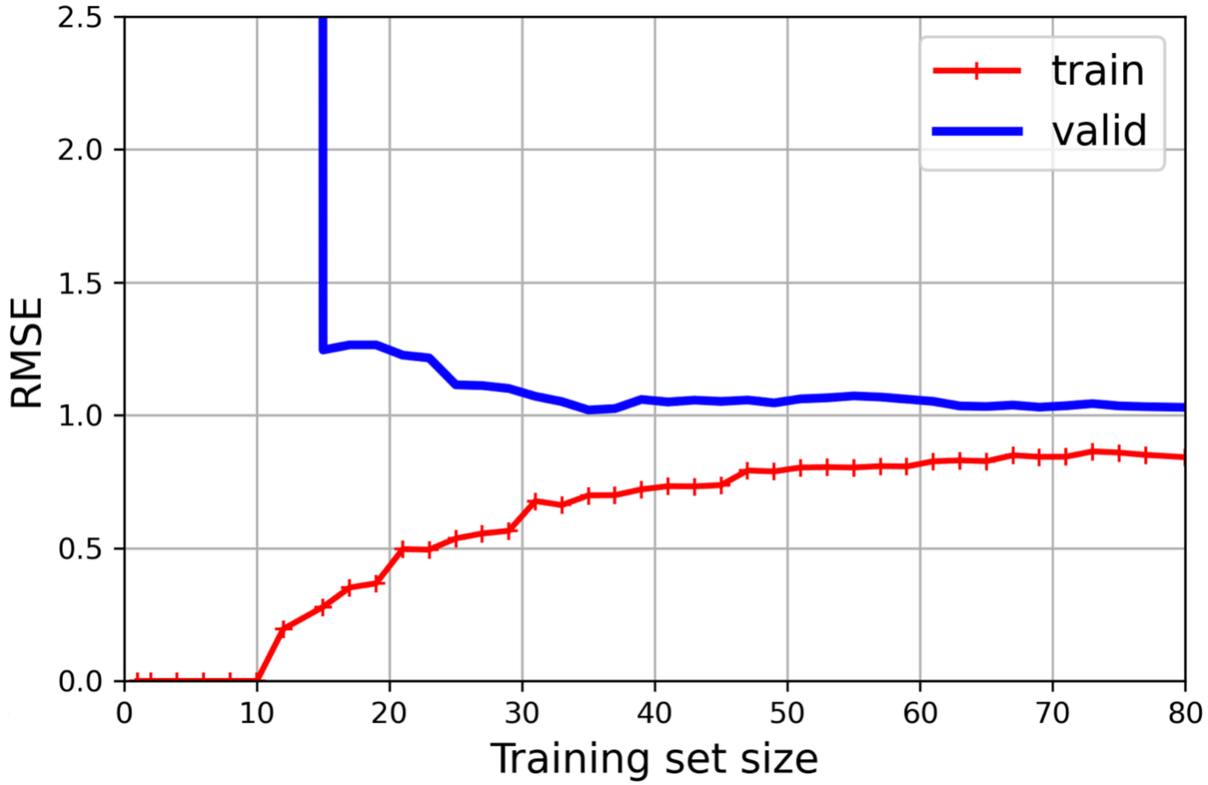
Difference?

Difference?

Linear



Polynomial — degree 10



Generalization error

- Bias
  - Due to wrong assumptions
    - Assuming that the data is linear when it is actually quadratic
  - A high-bias model is most likely to underfit the training data
- Variance
  - Excessive sensitivity to small variations in the training data
  - High variance —> overfitting

Generalization error

- Irreducible error
  - Noisiness of the data
  - Clean up the data
- Tradeoff
  - Increasing model's complexity
    - Decrease bias and increase variance
  - Vice-versa

Linear model regularisation

- Polynomial models
  - We can reduce the degree
- Linear models
  - Constraining the weights of the mode
    - Ridge Regression
    - Lasso Regression
    - Elastic Net

Artificial Intelligence and Machine Learning for Networks and IoT

Class 2 — Linear Regression

Pedro Braconnot Velloso