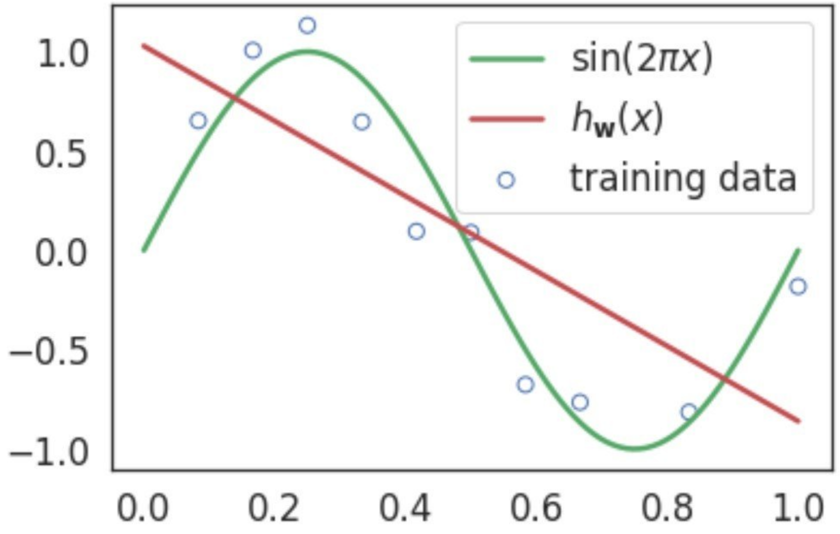
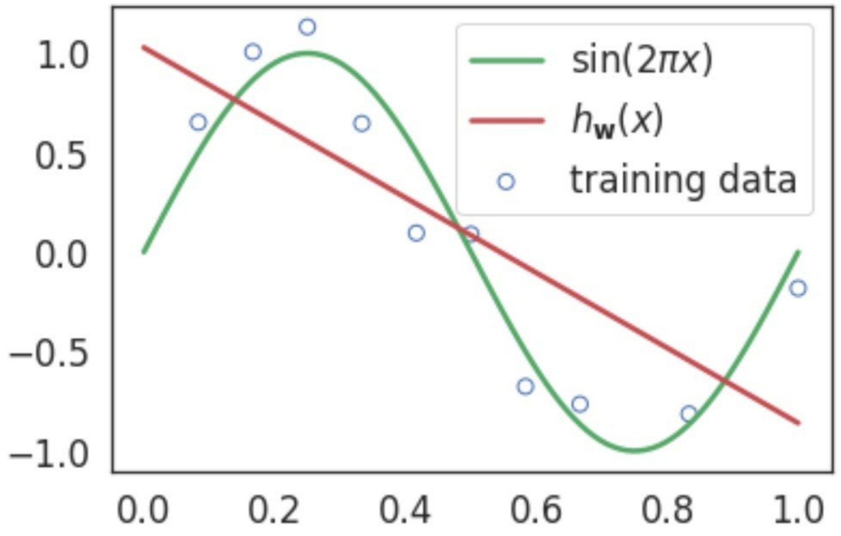
Part II: Polynomial Regression

Why polynomial regression?



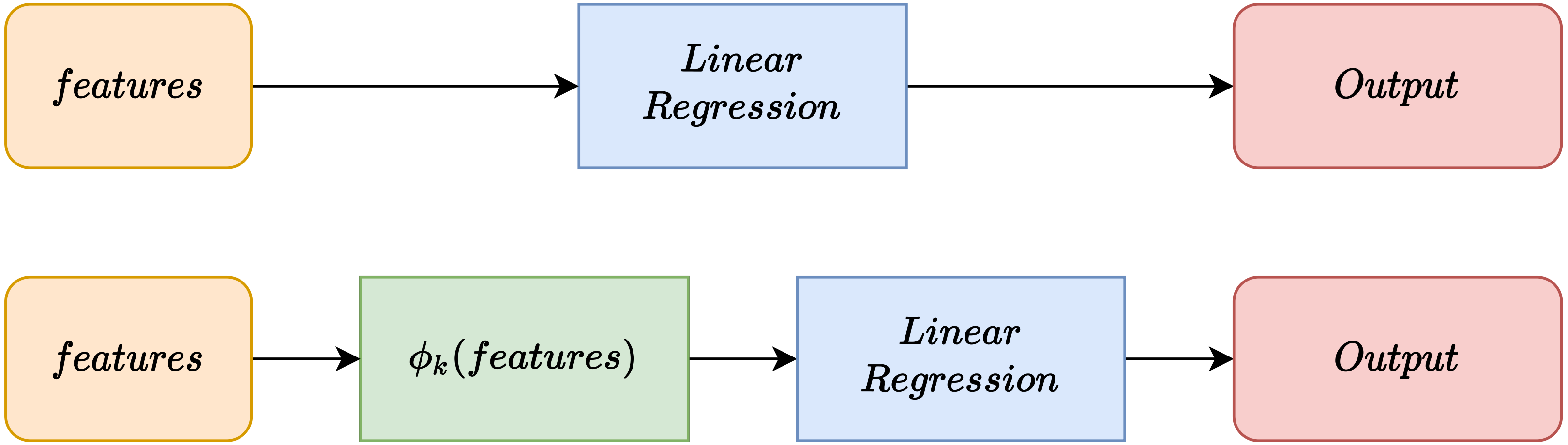
Many times the relationship between the input features and the output label is non-linear and simple linear models are not adequate to learn such mappings.

Polynomial regression: key idea



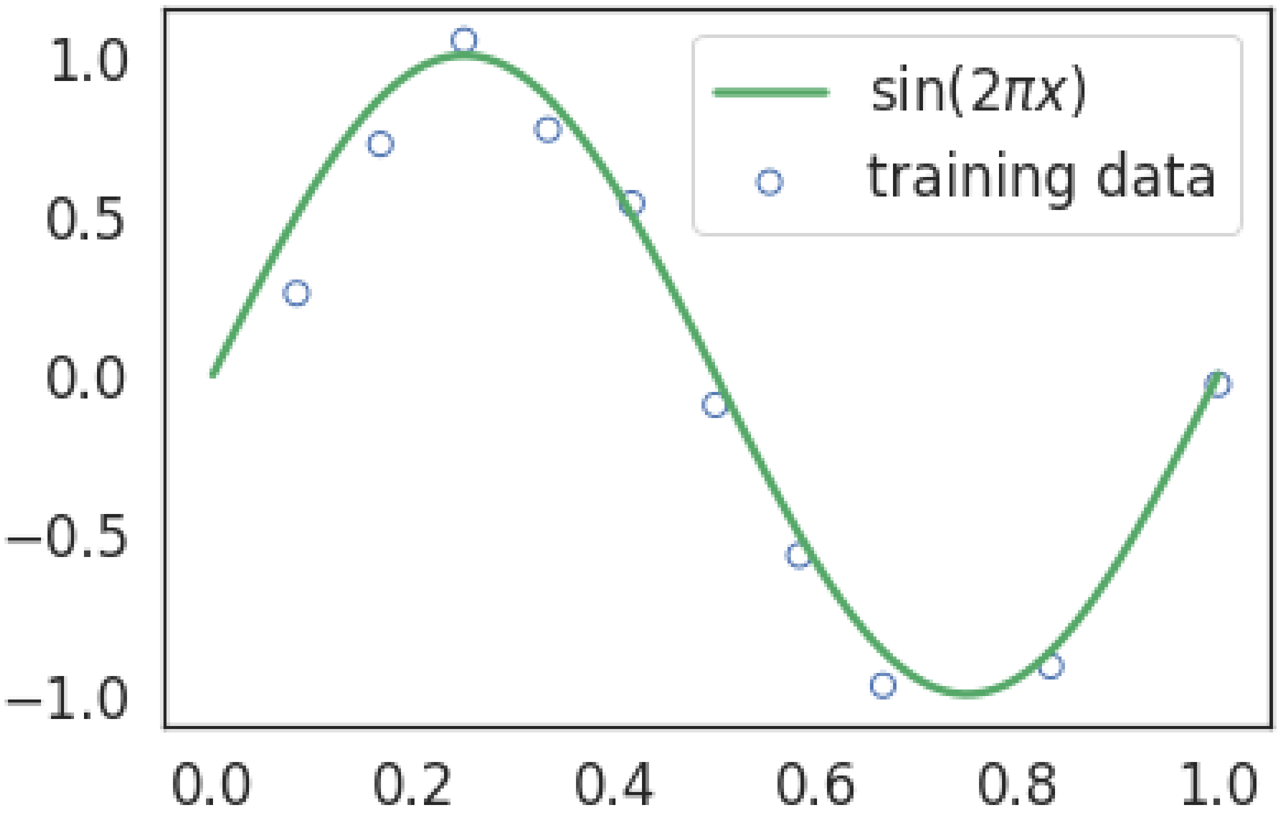
- The key idea is to create polynomial features by combining the existing input features.
- And then apply linear regression model on the polynomial feature representation.

Comparison: Linear and polynomial regression



ϕ_k(features) is called **polynomial transformation** of k-th order.

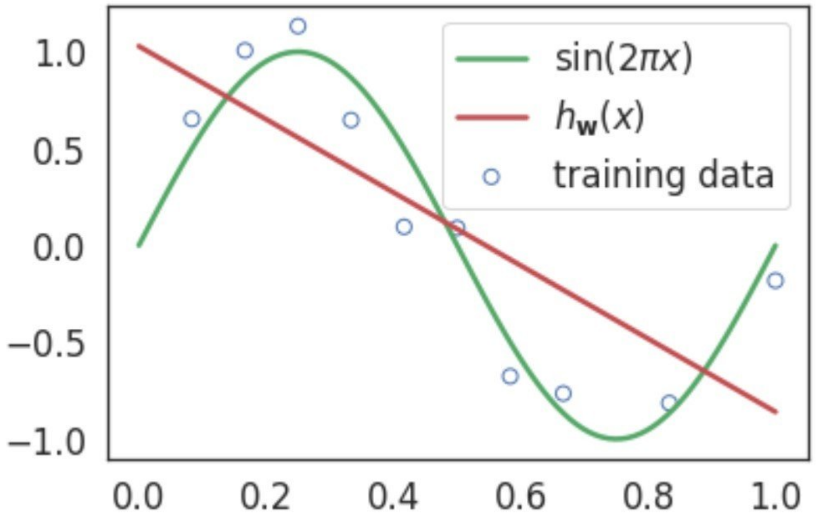
Polynomial regression: sample data



Why Polynomial Regression?

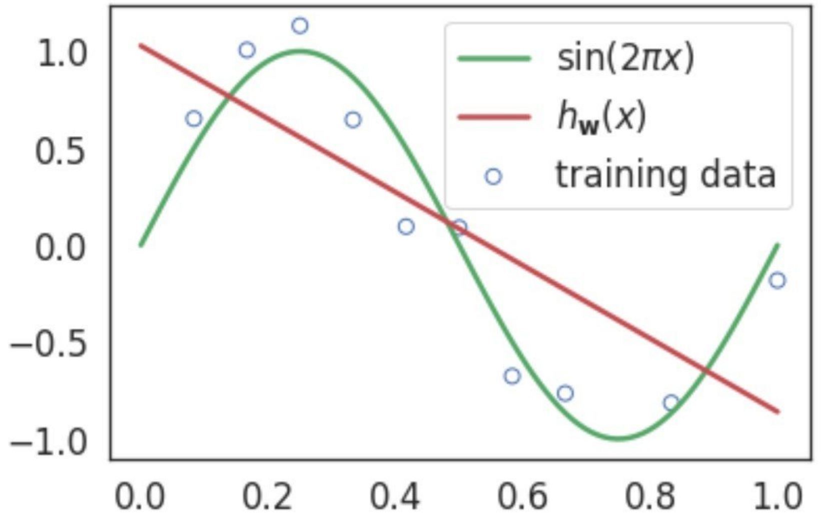
Linear regression without polynomial transformation.

$$h_{\mathbf{w}}(x) = w_0 + w_1 x$$



Clearly, this is not a great fit and we need to search for a better model.

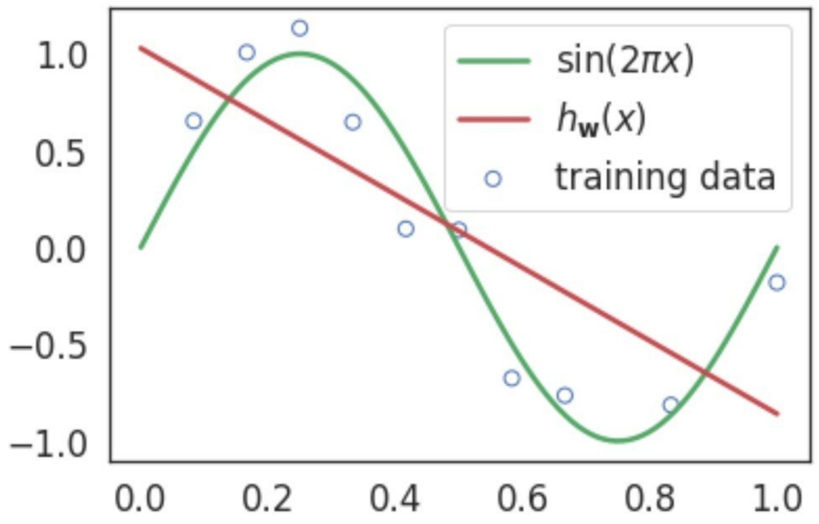
Fitness diagnostics



In this case, we are able to visualize the model fitment since we are dealing with data in 1D feature space.

As the number of features grow, it won't be possible for us to visualize the fitment in this manner.

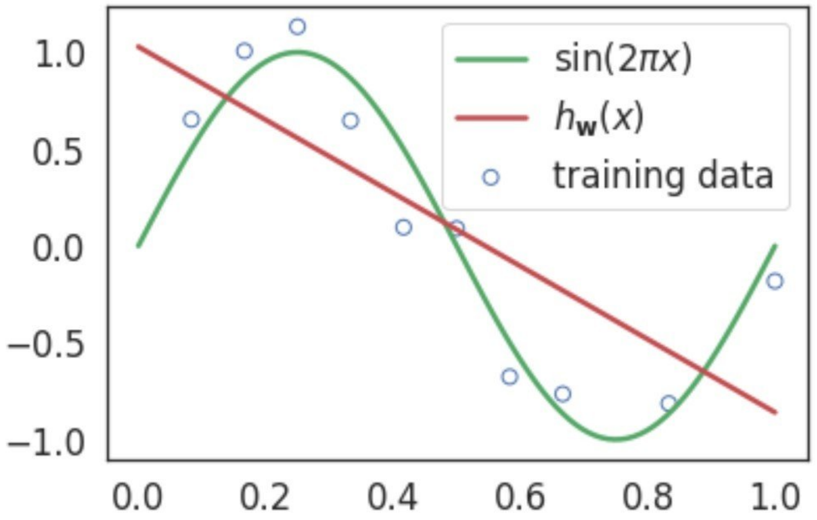
Fitness diagnostics

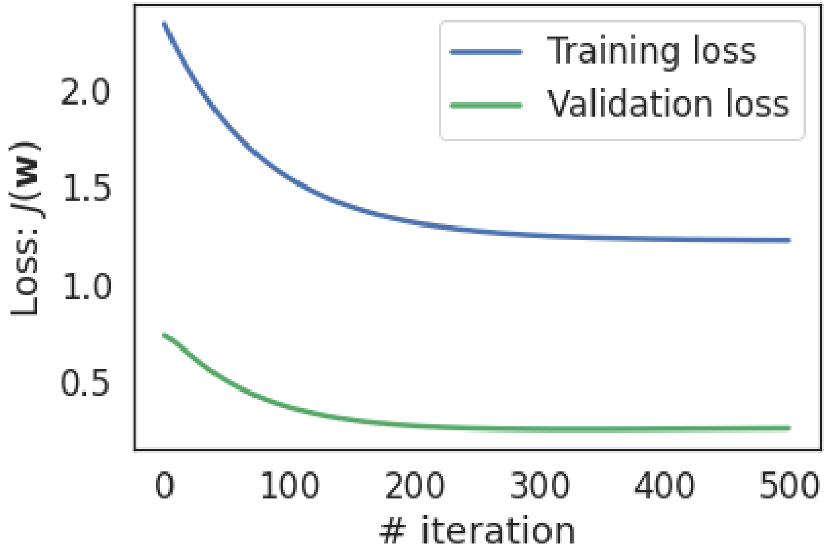


We rely on learning curve to determine quality of the fitment:

- Underfitting - both training and validation losses are high.
- Overfitting - training and validation losses decrease initially but then validation loss increases while training loss keeps decreasing.

Learning curve of linear regression





Linear model underfits: it is not enough to model the relationship between features and labels as present in the training data.

In this case, validation loss is less than the training loss (which is unusal), this is due to small amount of the validation data.

How can we fix underfitting?

- By increasing the capacity of the model to learn non-linear relationship between the features and the label.

- One way to achieve it is through polynomial transformation that constructs new features from the existing features.

Polynomial regression in single feature

For training data with a single feature x_1,

- First use k-th order polynomial transformation to create new features like $x^2, x^3, \ldots, x^k$ and
- Use linear regression model to learn relationship between x_1 and y.

$$\begin{aligned} y &= w_0 + w_1 x_1 + w_2 x_1^2 + \ldots + w_k x_1^k \\ &= w_0 1 + w_1 x_1 + w_2 x_1^2 + \ldots + w_k x_1^k \\ &= w_0 {x_1}^0 + w_1 x_1 + w_2 x_1^2 + \ldots + w_k x_1^k \\ &= \sum_{i=0}^{k} w_i x_1^i \end{aligned}$$

Note that the model is a non-linear function of x, but is a linear function of weight vector $\boldsymbol{w} = [w_0, w_1, \ldots, w_k]$.

Polynomial transformation

Let's represent this transformation in form of a vector ϕ with k components.

$$\phi = \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_k \end{bmatrix}$$

Each component denotes a specific transform to be applied to the input feature.

The polynomial regression model becomes:

$$y = \sum_{i=0}^{k} w_i (\phi(\mathbf{x}))_i$$

$$y = \mathbf{w}^T \phi(\mathbf{x})$$

Polynomial transformation: examples

- For a single feature x_1, $\phi_k = x_1^k$
- Two features (x_1, x_2), $\phi_2(x_1, x_2) = \left[1, x_1, x_2, x_1^2, x_2^2, x_1 x_2\right]$.
- Two features (x_1, x_2), $\phi_3(x_1, x_2) = \left[1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, x_2^3, x_1 x_2^2, x_1^2 x_2\right]$
- ϕ_3 contains features from ϕ_2.

Polynomial Transformation: Implementation

```python
import itertools
import functools

def polynomial_transform(x, degree):
  '''Performs transformation of input x into deg d polynomial features.

  Arguments:
    x: Data of shape (n,)
    degree: degree of polynomial

  Returns:
    Polynomial transformation of x
  '''
  if x.ndim == 1:
    x = x[:, None]

  x_t = x.transpose()
  features = [np.ones(len(x))]
  for degree in range(1, degree + 1):
    for items in itertools.combinations_with_replacement(x_t, degree):
      features.append(functools.reduce(lambda x, y: x * y, items))
  return np.asarray(features).transpose()
```

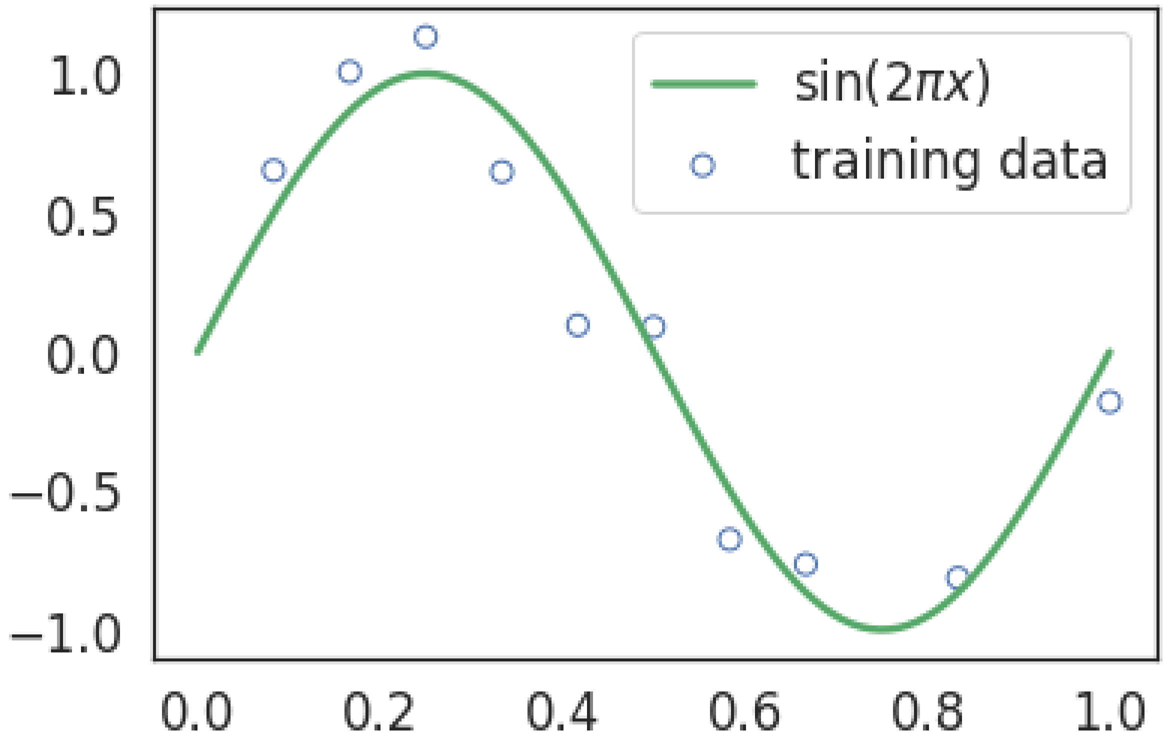
Polynomial transformation: implementation

An example:

```
polynomial_transform(np.array([[1, 2], [3, 4]]), degree=3)
```

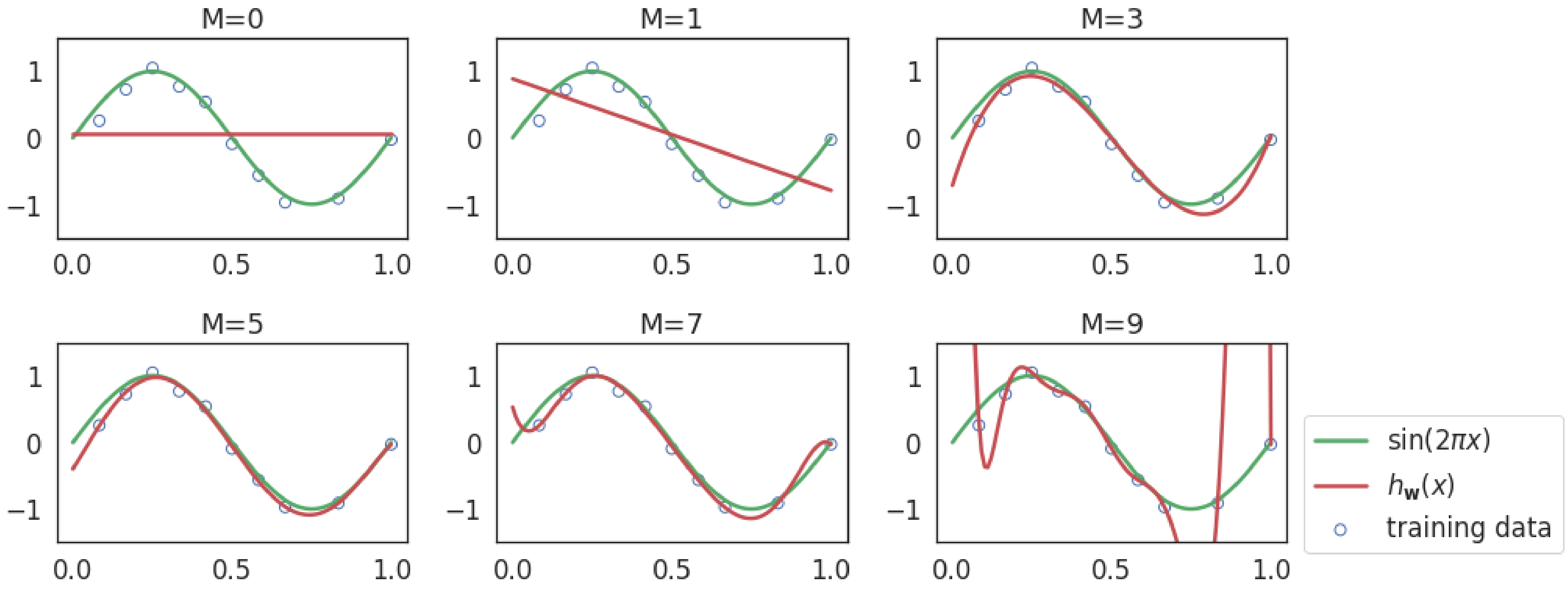
Output:

```
array([[ 1.,  1.,  2.,  1.,  2.,  4.,  1.,  2.,  4.,  8.],
       [ 1.,  3.,  4.,  9., 12., 16., 27., 36., 48., 64.]])
```

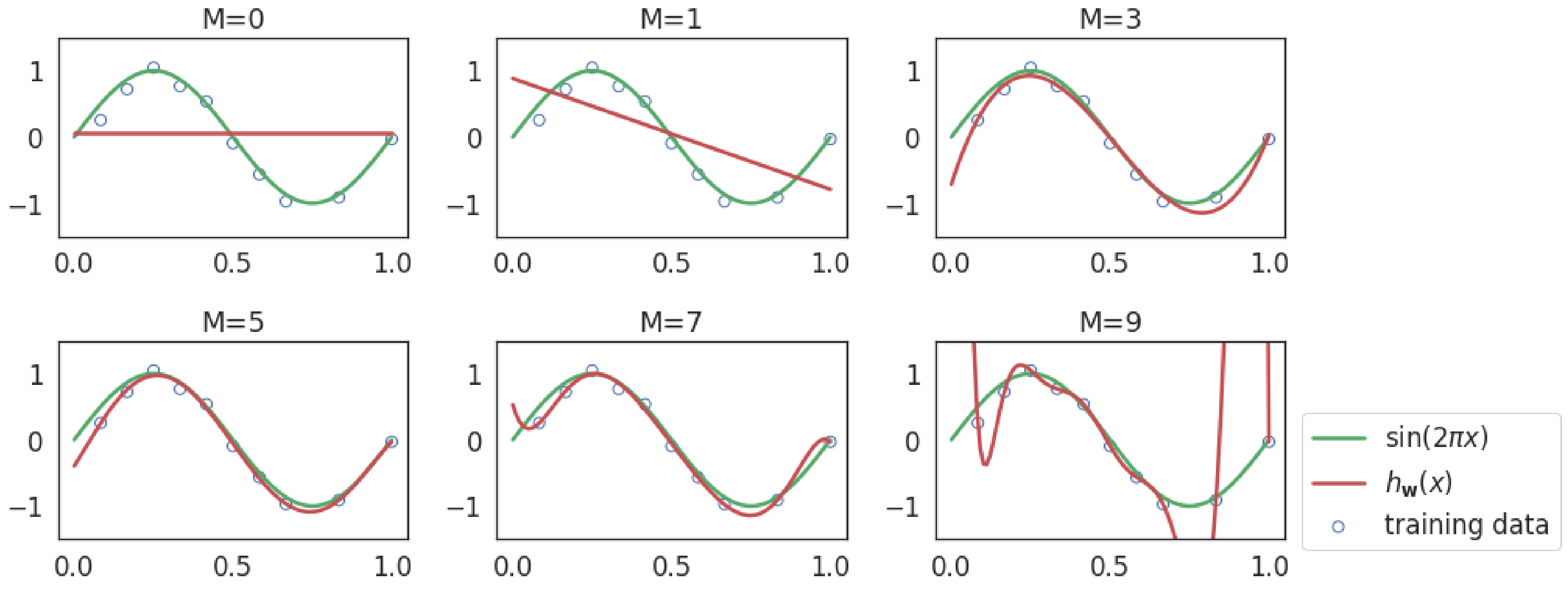


Let's fit a polynomial model of different orders $k = \{0, 1, 2, \ldots, 9\}$ on this data.

Fitting Polynomial Regression Models



Fitting Polynomial Regression Models



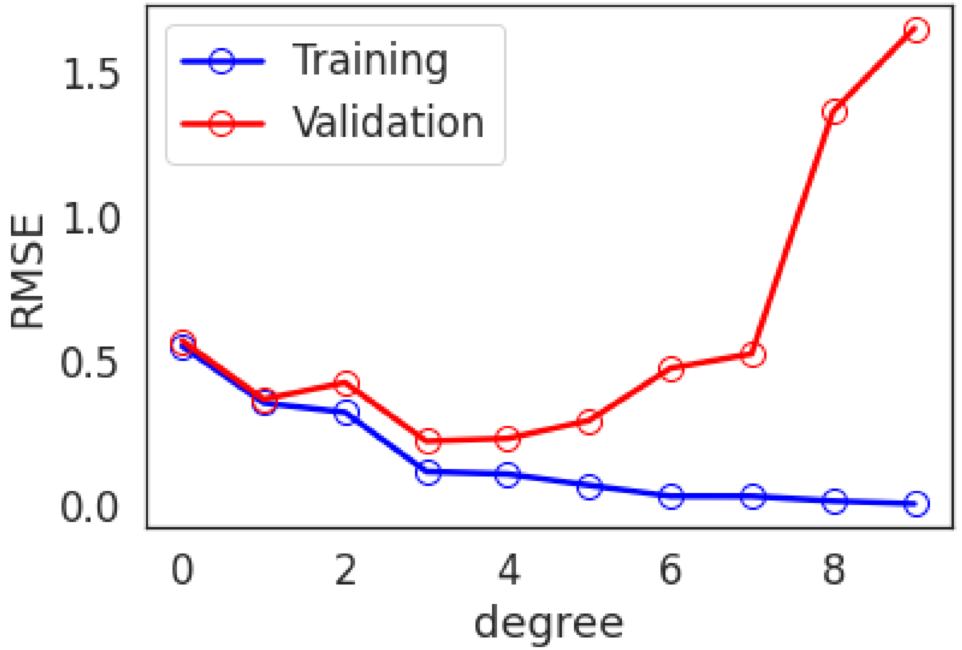
- Lower degree (0 and 1) polynomial model: Underfitting.
- Third order (degree=3) polynomial model: Reasonable fit.
- Higher degree ≥ 7 polynomial model: Overfitting.

Model diagnostics: learning curves

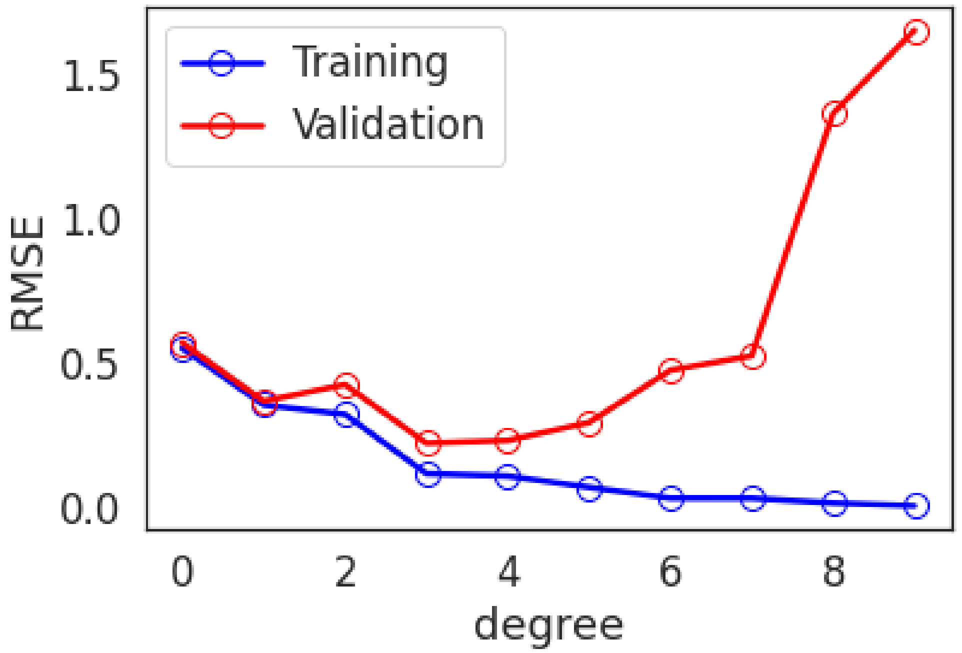
(1) Train polynomial model of degrees: $\{0, 1, \ldots, k\}$

(2) Calculate training and validation RMSE.

(3) Plot degree vs. RMSE plot.



Model diagnostics: learning curves



- The training and validation errors are close until certain degree.
- After a point, the training error continues to reduce, while the validation error keeps increasing.
- RMSE increases sharply for degrees $\geq$ 7. This is a signature of overfitting.

Issues with polynomial regression

- Higher order polynomial models are very flexible, or in other words, they have higher capacity compared to lower order models.

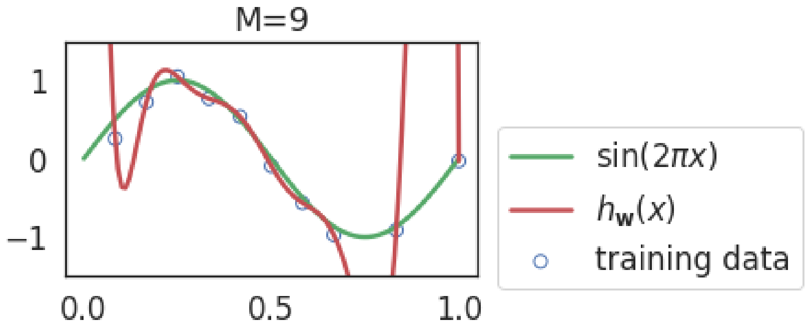
- Hence they are prone to overfitting compared to the lower degree polynomials.

- Perfect fit to training data, but poor prediction accuracy on validation data.

Demonstration of overfitting: deg = 9

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \end{bmatrix} = \begin{bmatrix} 1.76 \\ -115.74 \\ 2458.97 \\ -21739.10 \\ 103583.95 \\ -293329.12 \\ 507610.57 \\ -525717.33 \\ 298590.02 \\ -71343.99 \end{bmatrix}$$

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=0}^{j=9} w_j x^j$$

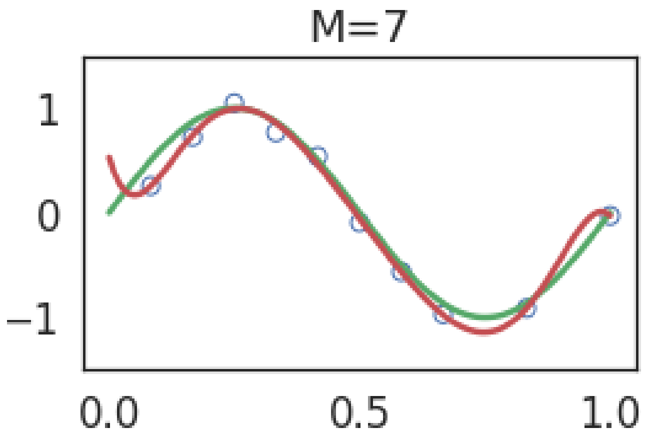


Observe that higher degree polynomial features have higher weights than others.

Demonstration of overfitting: deg = 7

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \end{bmatrix} = \begin{bmatrix} -4.23 \\ 99.31 \\ -641.24 \\ 1988.00 \\ -3332.88 \\ 2993.67 \\ -1304.04 \\ 201.43 \end{bmatrix}$$

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=0}^{j=7} w_j x^j$$



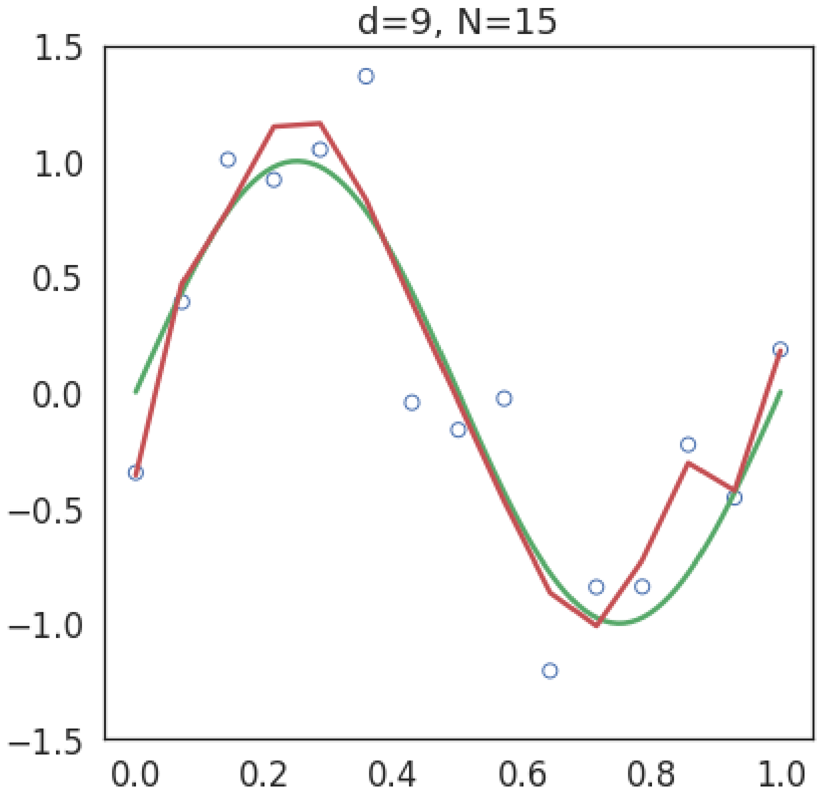
Observe that higher degree polynomial features have higher weights than others.

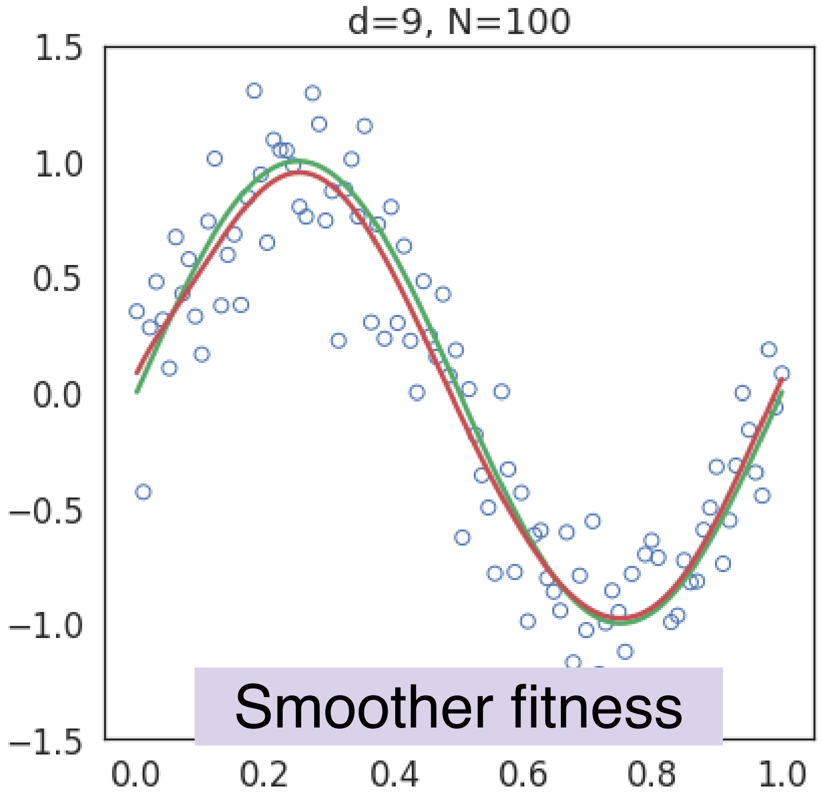
How to address overfitting?

- Using larger training sets.
- Controlling model complexity through regularization, which allows us to fit complex model on relatively smaller dataset without overfitting.

Fixing overfitting with more data

Train a polynomial model of degree 9 with two datasets - (i) with 15 points and (ii) with 100 points.





Fixing overfitting through regularization

Overfitting is caused by larger weights assigned to the higher order polynomial terms.

Regularization: Add a penalty term in the loss function for such large weights to control this behaviour.

At a high level, let's modifies loss function by adding a penalty term as follows:

$$J(\mathbf{w}) = \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{Y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{Y}\right) + \lambda\, \text{penalty}$$

Regularization modifies the loss function, which leads to the change in derivative of the loss function and hence a change in the weight update rule in gradient descent procedure.

Two components of regularization

$$J(\mathbf{w}) = \tfrac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{Y}\right)^T \left(\mathbf{X}\mathbf{w} - \mathbf{Y}\right) + \lambda\, \text{penalty}$$

- Penalty is a function of weight vector

- Regularization rate λ controls amount of penalty to be added to the loss function.

Regularization types

- L2 regularization (Ridge regression)
- L1 regularization (Lasso regression)
- Combination of L1 and L2: Elastic net regularization

Ridge Regression

Ridge regression uses second norm of the weight vector as a penalty term:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2 + \frac{\lambda}{2}\ \textstyle\sum_{i=1}^{m} w_i^2$$

This can be written in a vectorized form as follows

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\|\mathbf{w}\|_2 \\ &= \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \end{aligned}$$

Gradient calculation in ridge regression

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \left\{ \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T \right) (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left(\mathbf{w}^T \mathbf{X}^T - \mathbf{y}^T \right) (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) - \mathbf{y}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left(\mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} \right) - \left(\mathbf{y}^T \mathbf{X}\mathbf{w} - \mathbf{y}^T \mathbf{y} \right) + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{y}^T \mathbf{y} + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} + \lambda \mathbf{w}^T \mathbf{w} \right\}
\end{aligned}
$$

These steps are same as gradient calculation in linear regression except for the penalty term.

Gradient calculation in ridge regression

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{2}\frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{w}^T\mathbf{X}^T\mathbf{w}\mathbf{X} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y} + \lambda\mathbf{w}^T\mathbf{w}\right\} \\
&= \frac{1}{2}\left\{\frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{w}^T\mathbf{X}^T\mathbf{w}\mathbf{X}\right\} - \frac{\partial}{\partial \mathbf{w}}\left\{2\mathbf{w}^T\mathbf{X}^T\mathbf{y}\right\} + \frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{y}^T\mathbf{y}\right\} + \frac{\partial}{\partial \mathbf{w}}\left\{\lambda\mathbf{w}^T\mathbf{w}\right\}\right\} \\
&= \frac{1}{2}\left\{2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 0 + 2\lambda\mathbf{w}\right\} \\
&= \frac{1}{2}\left\{2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{w}\right\} \\
&= \frac{1}{2} \times 2 \times \left\{\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + \lambda\mathbf{w}\right\} \\
&= \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + \lambda\mathbf{w}
\end{aligned}
$$

Normal Equation

Let's set $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ to 0 and solving for $\mathbf{w}$:

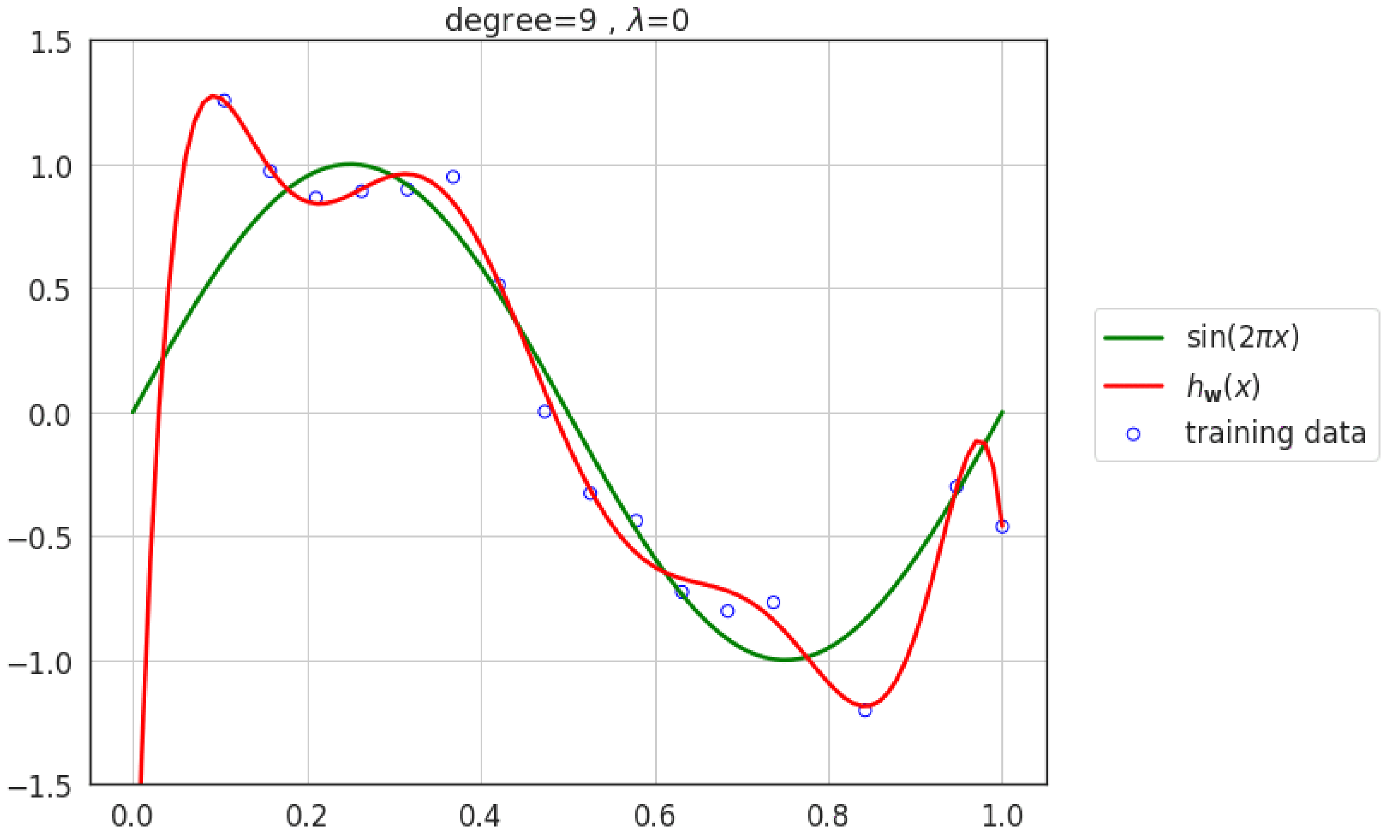
$$\begin{aligned} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + \lambda\mathbf{w} = 0 \\ \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\ \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\ \mathbf{w} &= \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y} \end{aligned}$$

- [No regularization] When $\lambda \to 0$, $\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$ (Same as the linear regression solution)
- [Infinite regularization] When $\lambda \to \infty$, $\mathbf{w} = 0$

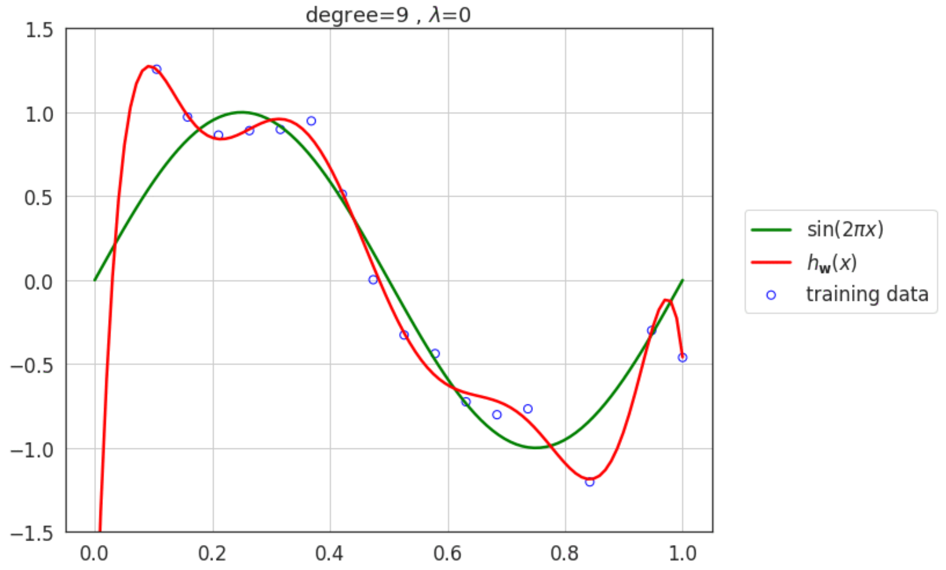
Gradient Descent Update Rule

$$\begin{aligned}
\mathbf{w}_{k+1} &:= \mathbf{w}_k - \alpha \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \\
&:= \mathbf{w}_k - \alpha \left(-\mathbf{X}^T \mathbf{y} + \mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} \right) \\
&:= \mathbf{w}_k - \alpha \left(\mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{X}^T \mathbf{y} + \lambda \mathbf{w} \right) \\
&:= \mathbf{w}_k - \alpha \left\{ \mathbf{X}^T \left(\mathbf{X} \mathbf{w} - \mathbf{y} \right) + \lambda \mathbf{w} \right\}
\end{aligned}$$

Effect of regularization λ



Effect of regularization λ

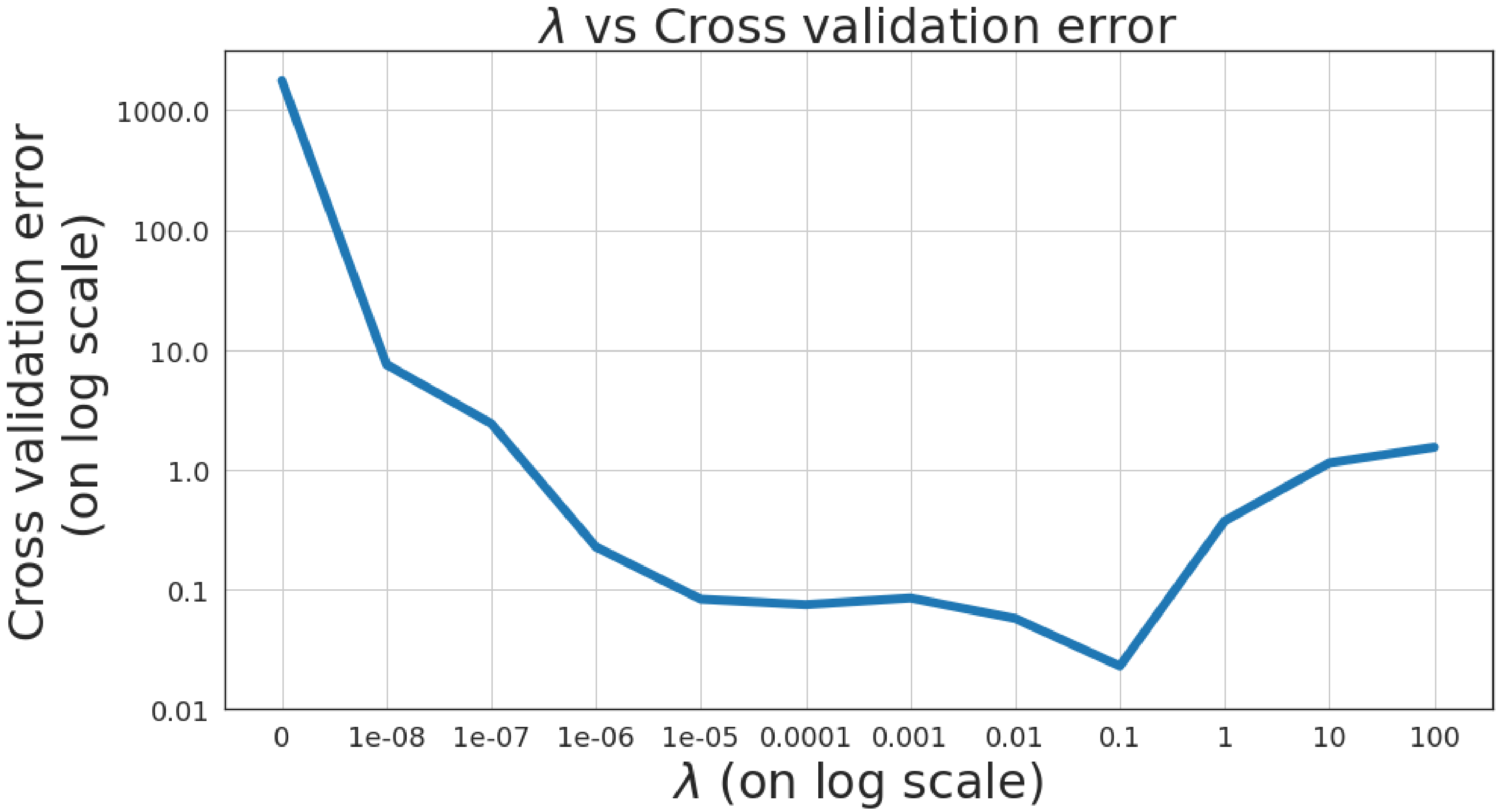


- As value of λ increases, the model loses its capacity and eventually for very large values of λ, the model underfits the data.

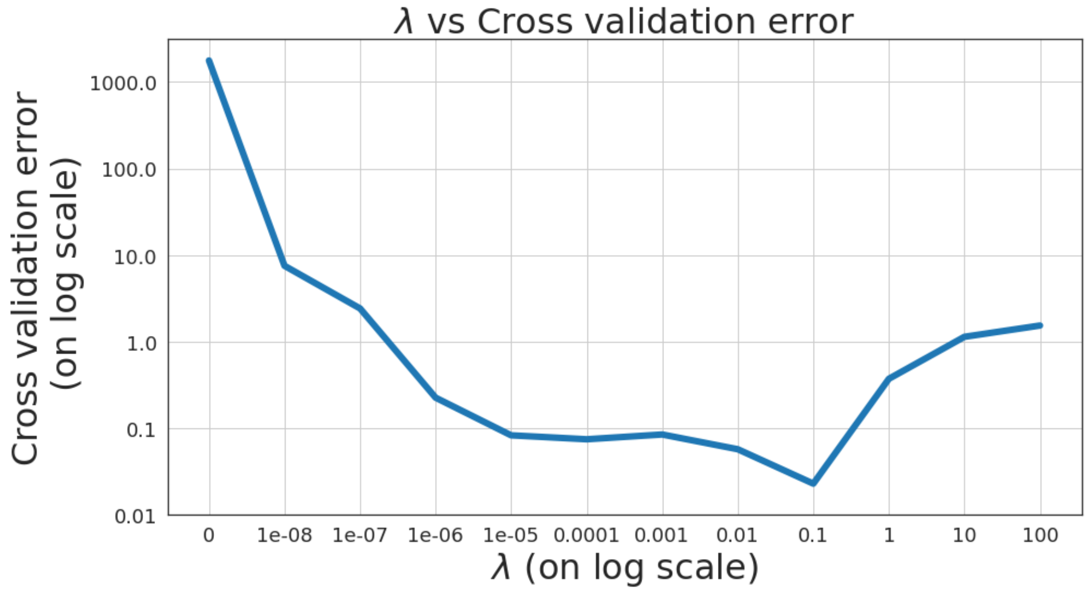
Choosing λ

1. Construct a set of values of λ that we want to experiment with.
2. For each candidate value of λ:
    1. Train the model and calculate cross validation error on validation set.
    2. Choose the the value of λ which results in the least cross validation error.
3. With the chosen value of λ, train the model on entire training set.
4. Report the model performance on the test set.

λ vs Cross validation error

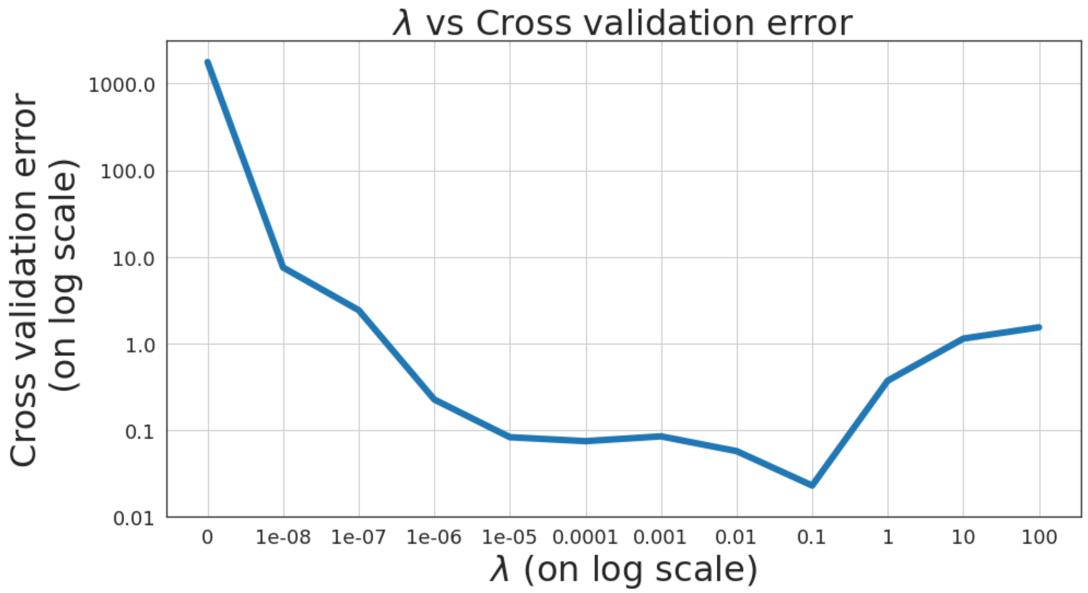


λ vs Cross validation error



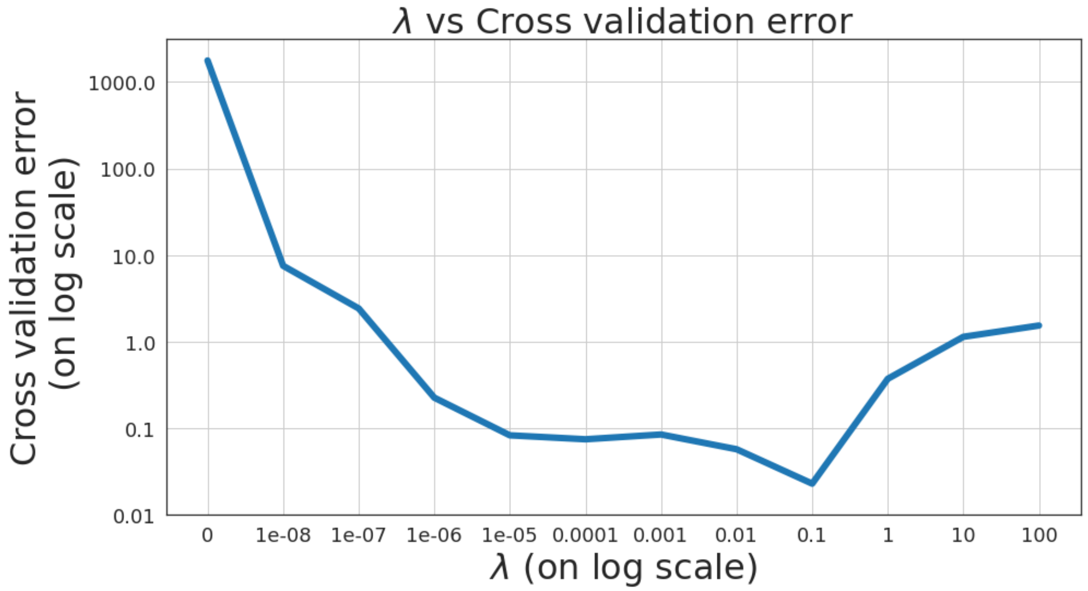
Typically above chart looks like a bowl shape, however, due to small number of features and samples, the chart takes shape like above.

λ vs Cross validation error



The most appropriate value of λ results in lowest cross validation error.

λ vs Cross validation error



Most appropriate value of λ from above figure is 0.1.

Lasso Regression

$$
\begin{aligned}
J(\mathbf{w}) &= \frac{1}{2}\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right) + \frac{\lambda}{2}\|\mathbf{w}\|_1 \\
&= \frac{1}{2}\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right) + \frac{\lambda}{2}\sum_{j=1}^{m}|w_j|
\end{aligned}
$$

We need specialized optimization algorithms to estimate weight vector in lasso regression, which are beyond scope of this course.

We will use sklearn implementation of Lasso in ML practice course.

Why does ridge and lasso work?