Day 4: Linear Classifiers

Summer STEM: Machine Learning

Department of Electrical and Computer Engineering
NYU Tandon School of Engineering
Brooklyn, New York

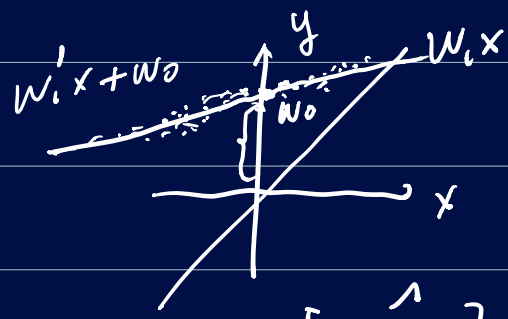
June 25, 2020

Linear Regression

- scalar-valued features x_i

$$\hat{y}_i = w_1 x_i + w_0$$

w_0 : intercept / bias



- Loss : $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

- matrix-vector form

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = Xw$$

$$= \begin{bmatrix} w_0 + w_1 x_1 \\ \vdots \\ w_0 + w_1 x_N \end{bmatrix}$$

Alternatively, we can write

$$\hat{Y} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} [w_1] + \left\{ \begin{bmatrix} w_0 \\ w_0 \\ \vdots \\ w_0 \end{bmatrix} \right. \text{intercept vector}$$

.fit(X , y)

↑ design matrix

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{bmatrix} = Xw$$

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_N \end{bmatrix} \qquad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\| Y - \hat{Y} \|^2 = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$= \sum_{i=1}^{N} \left(y_i - (w_1 x_i + w_0) \right)^2$$

- Matrix multiplication

$$(AB)_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj}$$

The element of the product at the i-th row and the j-th column is the inner product of the i-th row of A and j-th column of B.

- Inner product (dot product) of vectors.

Given two vectors u, v of the same size D

$$u = (u_1, u_2, \ldots, u_D)$$

$$v = (v_1, v_2, \ldots v_D)$$

$$u \cdot v = u_1 v_1 + u_2 v_2 + \cdots + u_D v_D$$

$$= \sum_{j=1}^{D} u_j v_j$$

Ex: $u = (1, 2, 3)$

$v = (4, 5, 6)$

$u \cdot v = 1 \times 4 + 2 \times 5 + 3 \times 6$

$= 4 + 10 + 18$

$= 32$ a scalar

Ex. matrix - multiplication

$$A = \begin{bmatrix} 3 & 1 \\ 2 & 4 \\ 5 & 7 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & -2 \\ 0 & 3 \end{bmatrix}$$

$(AB)_{11} = 3$

$(AB)_{21} = 2$

AB is $(3, 2)$

$$\|Y - \hat{Y}\|^2 = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad \hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_N \end{bmatrix}$$

$$\|Y - \hat{Y}\|^2 = \left\| \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ \vdots \\ y_N - \hat{y}_N \end{bmatrix} \right\|^2$$

$$= \left(\sqrt{(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \cdots + (y_N - \hat{y}_N)^2} \right)^2$$

$$= (y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \cdots$$
$$+ (y_N - \hat{y}_N)^2$$

$$= \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$\hat{y}_i = w_0 + w_1 x_i$$

- $\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$

- $\frac{1}{N} \| Y - \hat{Y} \|^2$

where $\hat{Y} = Xw$

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ & \vdots \\ 1 & x_N \end{bmatrix} \qquad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

design matrix
(feature matrix)

- vector-valued features

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}$$

both of x_{i1} and x_{i2} are scalars

$$\hat{y}_i = w_0 + w_1 x_{i1} + w_2 x_{i2}$$

Design matrix in this case

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ & \vdots & \\ 1 & x_{N1} & x_{N2} \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_N \end{bmatrix} = X w$$

↑ design matrix

for x_i of size D

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iD} \end{bmatrix}$$

the design matrix is

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1D} \\ 1 & x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & & & & \\ 1 & x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$$\hat{y} = Xw \qquad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{bmatrix}$$

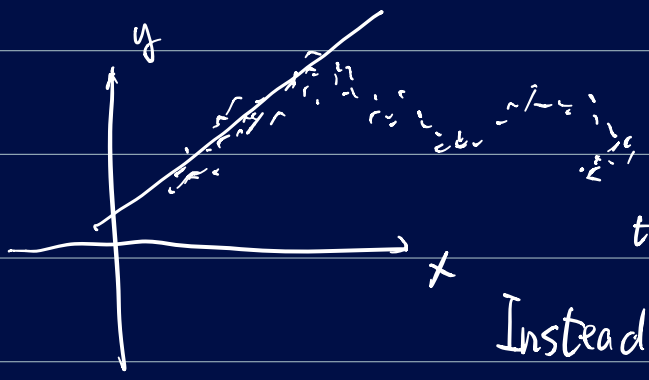
$$\text{Loss} : \frac{1}{N} \| y - \hat{y} \|^2$$

$$= \frac{1}{N} \| y - Xw \|^2$$

Our task is to find optimal parameters w^* such that the loss $\frac{1}{N}\|Y - Xw^*\|^2$ is minimized (has the least value for all possible w)

For linear regression with MSE loss

$$w^* = (X^T X)^{-1} X^T y$$



$\hat{y} = w_0 + w_1 x$ is not approapriate to fit this dataset

Instead, use polynomials

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M$$

- p-norm `np.linalg.norm(u, ord=2)`

Given a vector $u = \begin{bmatrix} u_1 \\ \vdots \\ u_D \end{bmatrix}$

$$\|u\| = \sqrt{|u_1|^2 + |u_2|^2 + \cdots |u_D|^2} \quad (L2\text{-norm})$$

$$\|u\|_p = (|u_1|^p + |u_2|^p + \cdots + |u_D|^p)^{\frac{1}{p}}$$

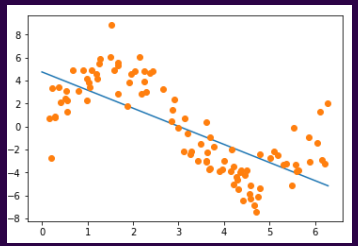
$$= \left(\sum_{j=1}^{D} |u_j|^p\right)^{\frac{1}{p}} \quad \left(2^{\frac{1}{2}} = \sqrt{2} \quad 2^{\frac{1}{3}} = \sqrt[3]{2}\right)$$

L1 norm $|u_1| + |u_2| + \cdots + |u_D|$

(L0 norm $\max_j |u_j|$)

Polynomial Fitting

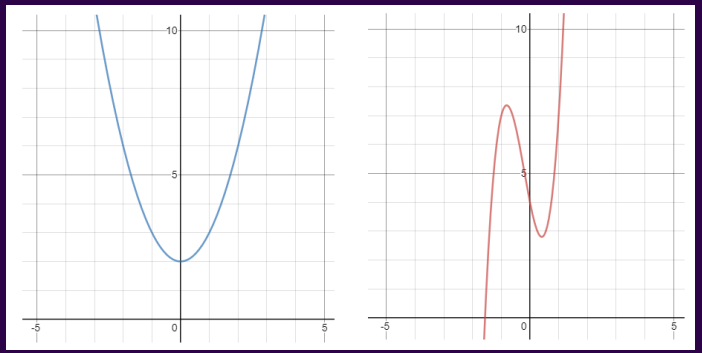
- We have been using straight lines to fit our data. But it doesn't work well every time
- Some data have more complex relation that cannot be fitted well using a straight line



- Can we use some other model to fit this data?

Polynomial Fitting

- Can we use a polynomial to fit our data?
- Polynomial: A sum of different powers of a variable
  - Examples: $y = x^2 + 2$, $y = 5x^3 - 3x^2 + 4$



Polynomial Fitting

- Polynomials of x: $\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_M x^M$
- M is called the order of the polynomial.
- The process of fitting a polynomial is similar to linearly fitting multivariate data.

$$\hat{y}_1 = w_0 + w_1 x_1 + w_2 x_1^2 + \cdots + w_M x_1^M$$

$$\hat{y}_2 = w_0 + w_1 x_2 + w_2 x_2^2 + \cdots + w_M x_2^M$$

$$\vdots$$

$$\hat{y}_N = w_0 + w_1 x_N + w_2 x_N^2 + \cdots + w_M x_2^M$$

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{bmatrix} \qquad X = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^M \\ 1 & x_2 & x_2^2 & \cdots & x_2^M \\ & & \vdots & & \\ 1 & x_N & x_N^2 & \cdots & x_N^M \end{bmatrix}$$

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} \qquad \hat{Y} = Xw$$

MSE loss $J(w) = \frac{1}{N} \| Y - Xw \|^2$

$$w^* = (X^T X)^{-1} X^T Y$$

Originally, the features x_i is scalar-valued

x_i It is transformed to a

$\phi(x_i)$ ↓ vector-valued feature

$$[1 \quad x_i \quad x_i^2 \quad \cdots \quad x_i^M]^T$$

$$\hat{y} = w^T \phi(x) \leftarrow \text{linear model}$$

There are many different feature transformations

Ex. $\phi(x_i) \rightarrow [1 \; x_i \; x_i^2 \cdots x_i^M]^T$

element-wise: $\phi_j(x_i) = x_i^j$

↑ basis function

other possibilities for $\phi_j(x_i) = \exp\left(-\frac{(x_i - \mu_j)^2}{2s^2}\right)$

Polynomial fitting

- Rewrite in matrix-vector form

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \approx \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^M \\ 1 & x_2 & x_2^2 & \cdots & x_2^M \\ \vdots & & \ddots & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^M \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix}$$

- This can still be written as

  $Y \approx X\mathbf{w}$

- Loss $J(\mathbf{w}) = \frac{1}{N} \|Y - X\mathbf{w}\|^2$
- The i-th row of the design matrix X is simply a transformed feature $\phi(x_i) = (1, x_i, x_i^2, \cdots, x_i^M)$

Polynomial Fitting

- Original design matrix: $\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}$
- Design matrix after feature transformation:
$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^M \\ 1 & x_2 & x_2^2 & \cdots & x_2^M \\ \vdots & & \ddots & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^M \end{bmatrix}$
- For the polynomial fitting, we just added columns of features that are powers of the original feature

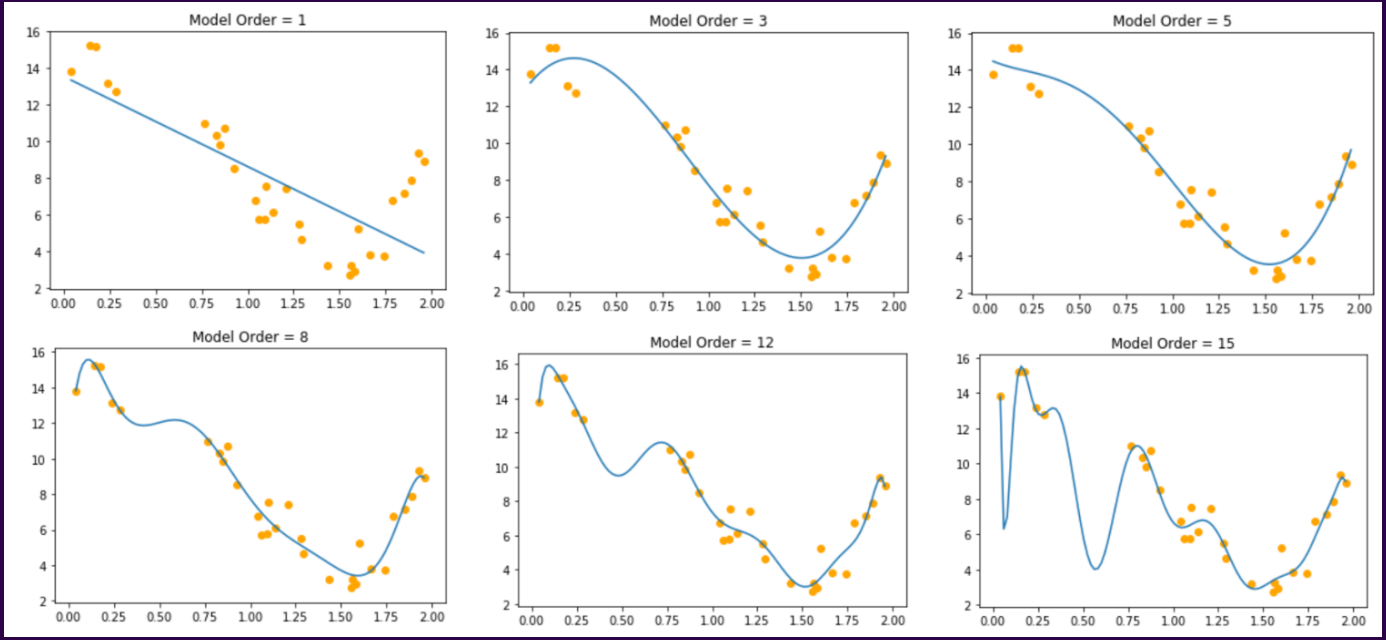
Linear Regression

- Model $\hat{y} = \mathbf{w}^T \phi(\mathbf{x})$
- Loss $J(\mathbf{w}) = \frac{1}{N} \|Y - X\mathbf{w}\|^2$
- Find $\mathbf{w}$ that minimizes $J(\mathbf{w})$

Overfitting

- We learned how to fit our data using polynomials of different order
- With a higher model order, we can fit the data with increasing accuracy
- As you increase the model order, at certain point it is possible find a model that fits your data perfectly (ie. zero error)
- What could be the problem?

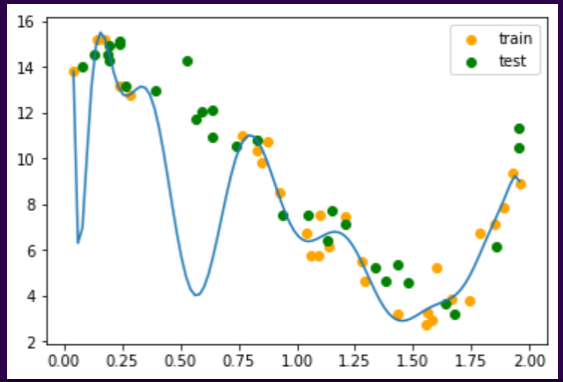
Overfitting



- Which of these model do you think is the best? Why?

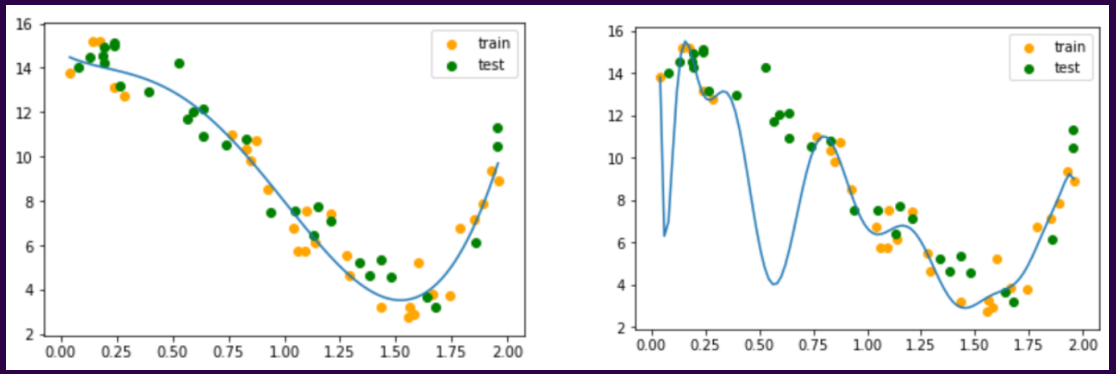
Overfitting

- The problem is that we are only fitting our model using data that is given
- Data usually contains noise
- When a model becomes too complex, it will start to fit the noise in the data
- What happens if we apply our model to predict some data that the model has never seen before? It will not work well.
- This is called over-fitting



Overfitting

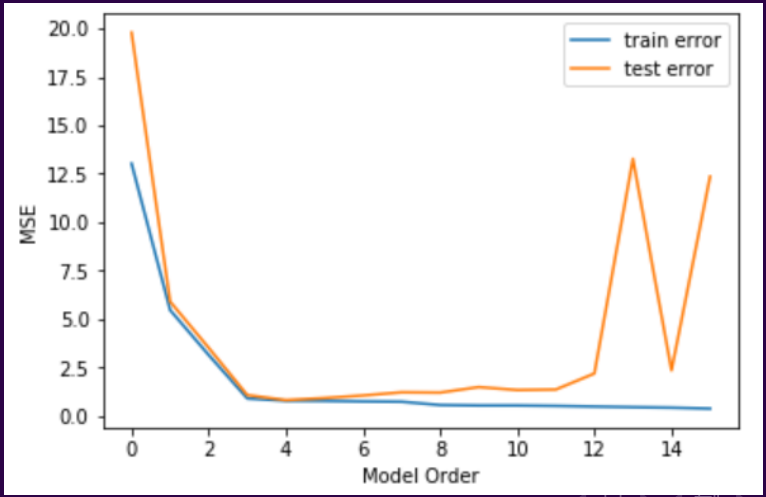
- Split the data set into a train set and a test set
- Train set will be used to train the model
- The test set will not be seen by the model during the training process
- Use test set to evaluate the model when a model is trained



- With the training and test sets shown, which one do you think is the better model now?

Train and Test Error

- Plot of train error and test error for different model order
- Initially both train and test error go down as model order increase
- But at a certain point, test error start to increase because of overfitting



model $\hat{y} = f(x; w)$

Loss MSE $J(w) = \frac{1}{N} \|Y - Xw\|^2$

Optimization (training) find optimal model parameters

Ex: for linear models $\hat{y} = w^T \phi(x)$

$$w^* = (X^T X)^{-1} X^T Y$$

.fit(X, Y) sklearn

always use only the training set.

Outline

How can we prevent overfitting without knowing the model order before-hand?

- **Regularization**: methods to prevent overfitting

How can we prevent overfitting without knowing the model order before-hand?

- **Regularization**: methods to prevent overfitting
  - We just covered regularization by model order selection

How can we prevent overfitting without knowing the model order before-hand?

- **Regularization**: methods to prevent overfitting
  - We just covered regularization by model order selection
- Is there another way? Talk among your classmates.

How can we prevent overfitting without knowing the model order before-hand?

- **Regularization**: methods to prevent overfitting
  - We just covered regularization by model order selection
- Is there another way? Talk among your classmates.
  - Solution: We can change our cost function.

Weight Based Regularization

- Looking back at the polynomial overfitting
- Notice that weight-size increases with overfitting

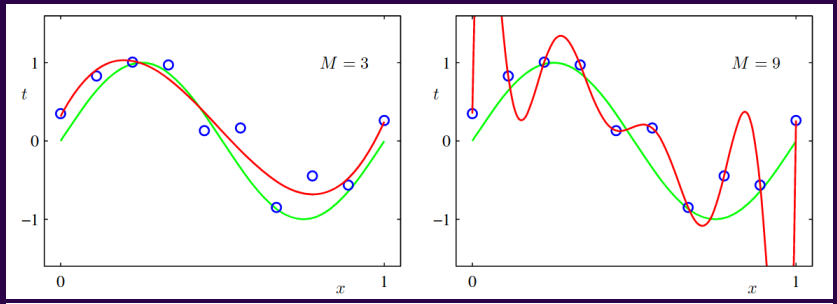


Table 1.1 Table of the coefficients $\mathbf{w}^\star$ for polynomials of various order. Observe how the typical magnitude of the coefficients increases dramatically as the order of the polynomial increases.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

NYU TANDON SCHOOL OF ENGINEERING

New Cost Function

$$J(\mathbf{w}) = \frac{1}{N}\|Y - X\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

$\lambda = 0$

λ very large

$\lambda\|w\|_1^2$

- Penalize complexity by simultaneously minimizing weight values.
- We call λ a **hyper-parameter**
  - λ determines relative importance

L2 norm reg. $\lambda\|w\|_2^2$ → small values of w_j

L1 norm $\lambda\|w\|_1^2$ → w with many elements being zero

Table 1.2 Table of the coefficients $\mathbf{w}^\star$ for $M = 9$ polynomials with various values for the regularization parameter λ. Note that $\ln\lambda = -\infty$ corresponds to a model with no regularization, i.e., to the graph at the bottom right in Figure 1.4. We see that, as the value of λ increases, the typical magnitude of the coefficients gets smaller.

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

NYU TANDON SCHOOL OF ENGINEERING

$$\hat{y}_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_D x_{iD}$$

what does it mean if some $w_j = 0$?

⟶ the prediction does not depend on the feature x_{ij}

Tuning Hyper-parameters

- Motivation: never determine a hyper-parameter based on training data
- **Hyper-Parameter**: a parameter of the algorithm that is not a model-parameter solved for in optimization.
  - Ex: λ weight regularization value vs. model weights ($\mathbf{w}$)
- Solution: split dataset into three
  - **Training set**: to compute the model-parameters ($\mathbf{w}$)
  - **Validation set**: to tune hyper-parameters (λ)
  - **Test set**: to compute the performance of the algorithm (MSE)

Machine Learning pipeline

1. Split your dataset (training/validation/test)
2. Chose a model ($\hat{y} = f(x; w)$)

   Ex: linear models $\hat{y} = w^T \phi(x)$
3. chose a loss

   Ex: MSE $J(w) = \frac{1}{N} \sum_{i=1}^{N} \| y_i - f(x_i; w) \|^2$
4. Training: find optimal w^* on the training set

   ↕ alternate until the validation error is the lowest

   tune hyper-parameters on the validation set
5. Test: Evaluate your model on the test set

$$w^* = (X_{train}^T X_{train})^{-1} X_{train}^T Y_{train}$$

matrix inverse is computationally expensive (→ very slow for large dataset)

ImageNet 14 Million

Outline

Motivation

- Cannot rely on closed form solutions
  - Computation efficiency: operations like inverting a matrix is not efficient
  - For more complex problems such as neural networks, a closed-form solution is not always available
- Need an optimization technique to find an optimal solution
  - Machine learning practitioners use **gradient**-based methods

Gradient Descent Algorithm

What is the gradient of a function?

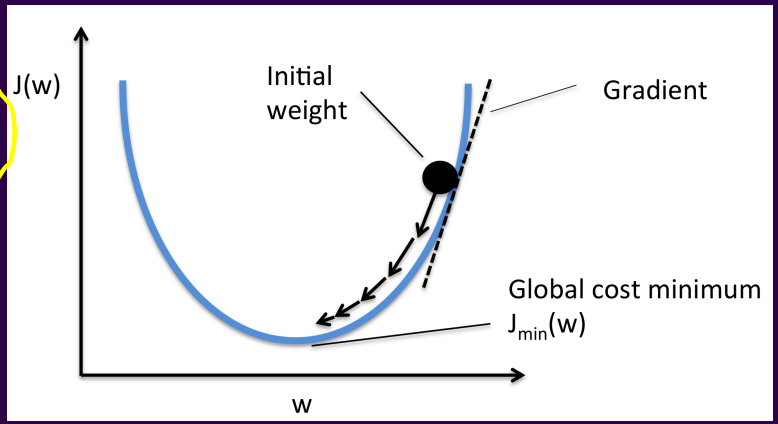
- Update Rule

  Repeat{

  $$\mathbf{w}_{new} = \mathbf{w} - \alpha \nabla_{\mathbf{w}} J$$

  }

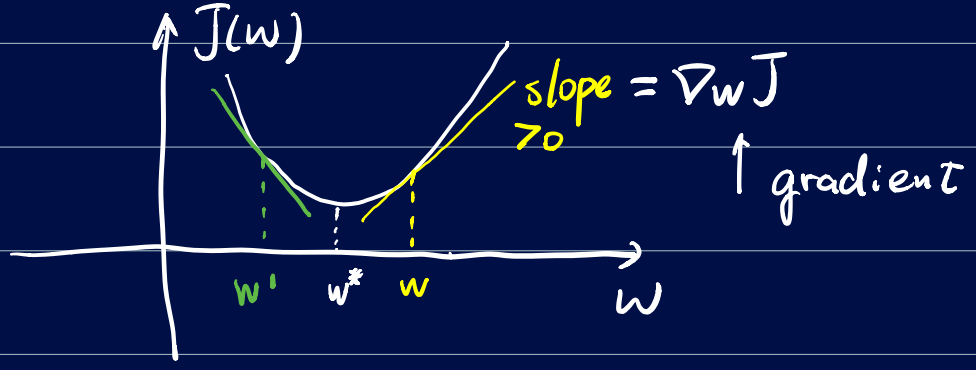
  α is the learning rate



Toy example $\hat{y} = wx$

$$J(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

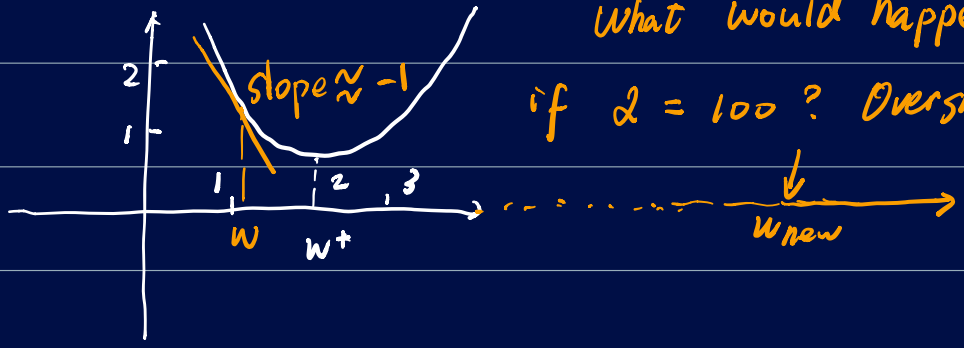
$$= \frac{1}{N}\sum_{i=1}^{N}(y_i - w x_i)^2$$



Slope >0 w should decrease

Slope <0 w should increase

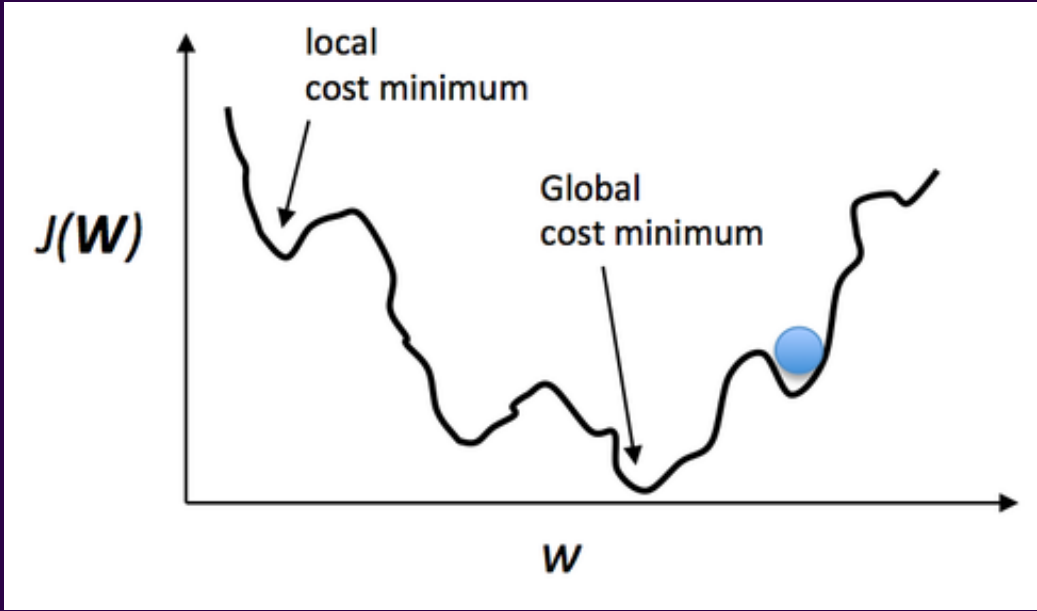
Update rule $w_{new} = w - \alpha \nabla_w J$



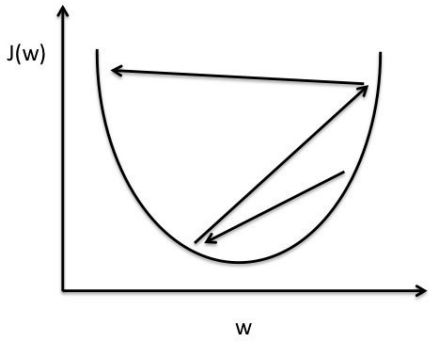
What would happen if $\alpha = 100$? Overshoot!

General Loss Function Contours

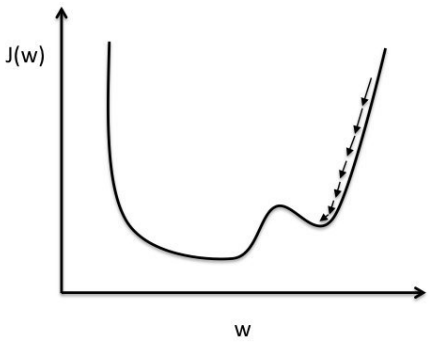
- Most loss function contours are not perfectly parabolic
- Our goal is to find a solution that is very close to global minimum by the right choice of hyper-parameters



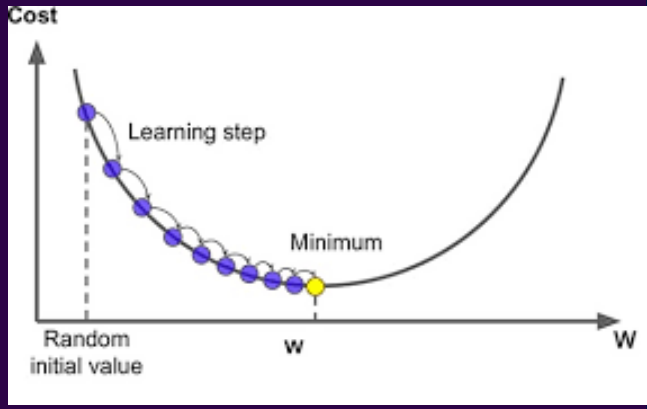
Understanding Learning Rate



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.



Correct learning rate

Some Animations

- Demonstrate gradient descent animation