

CHL7001H S1 Applied Deep Learning

Lecture 2: Fundamental ML theory and concepts

Supervised VS unsupervised

MACHINE LEARNING	CONTINUOUS	CATEGORICAL
SUPERVISED	REGRESSION (e.g. stock price)	CLASSIFICATION (e.g. customer churn)
UNSUPERVISED	DIMENSION REDUCTION	CLUSTERING

Understanding different Machine Learning models

It only takes 3 common components to understand **different models**:

1. **The hypothesis** of the relationship between output and the input data: a linear relationship, a tree structure, or a network?
2. **The cost (loss) function**: a formula to calculate how close our prediction is to the correct target.
3. **The optimization process**: if we are not close enough, how can we change our model parameter to make our prediction better and better?

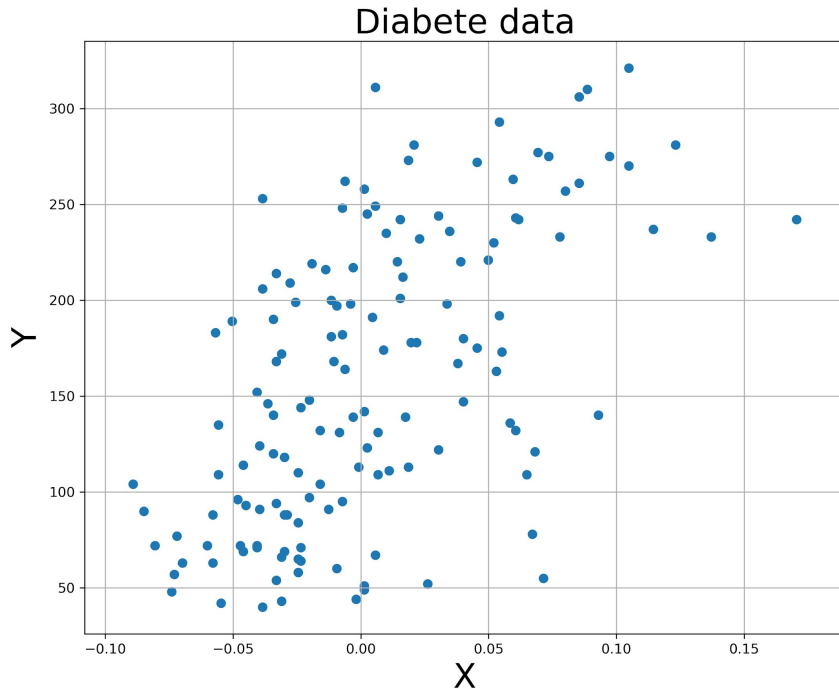
Regression

Simple linear regression uses a single predictor x

Given a dataset of m pairs (x^i, y^i) ,

The univariate linear regression model assumes that the output h is linear in the inputs x :

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

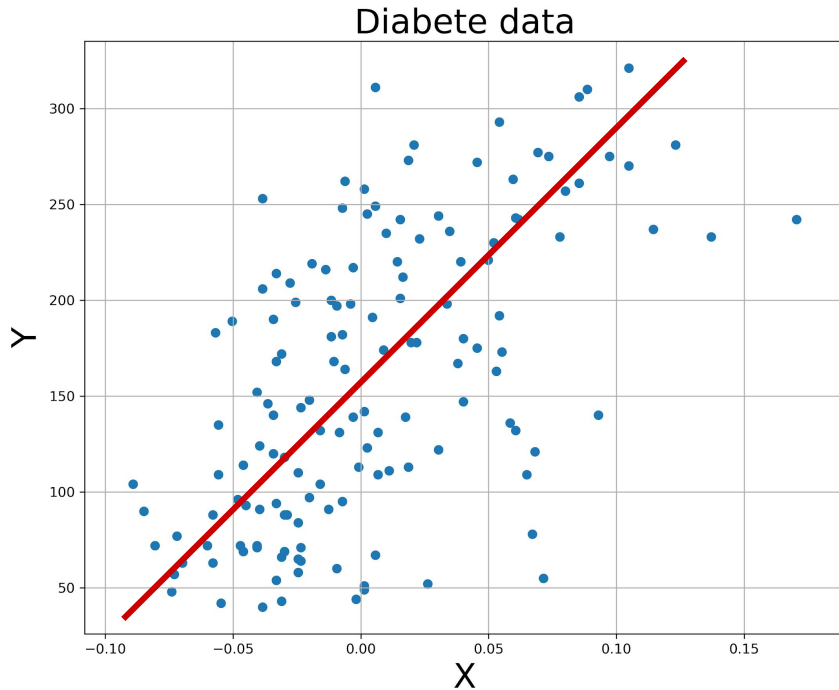


Simple linear regression uses a single predictor x

Given a dataset of m pairs (x^i, y^i) ,

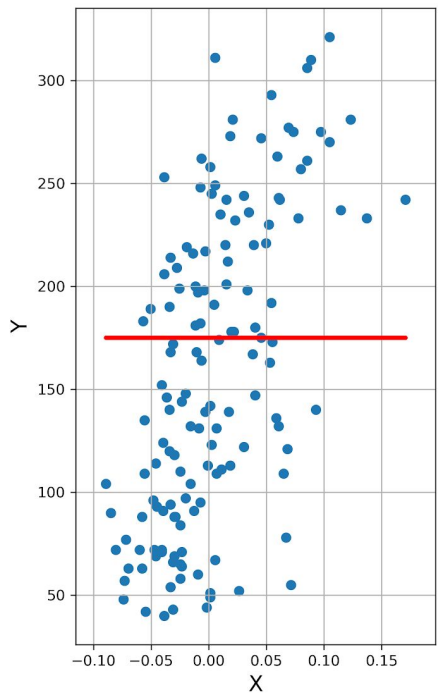
The univariate linear regression model assumes that the output h is linear in the inputs x :

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

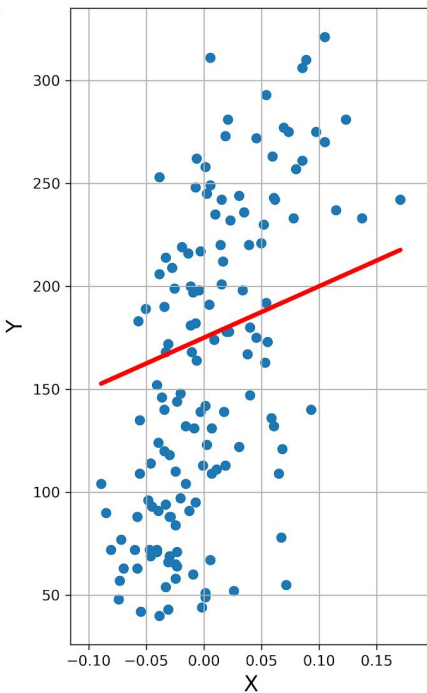


Fitting the line

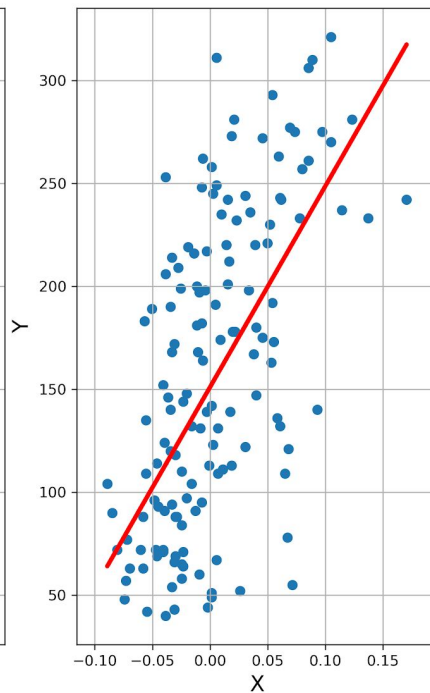
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



$$h(x) = 150$$



$$h(x) = 250x + 175$$



$$h(x) = 975x$$

How to find the best line???

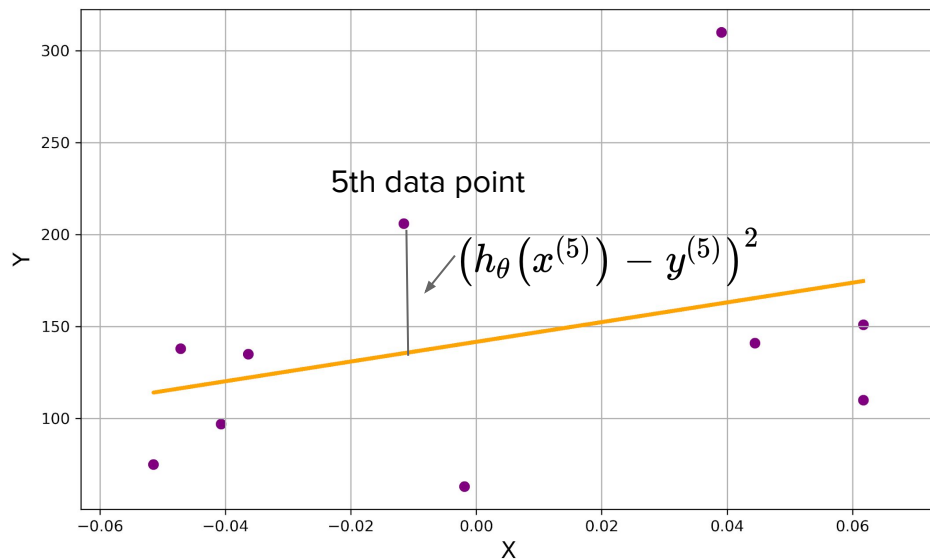


Optimization problem

How to pick the optimal θ_S ?

Cost function - squared errors

A formula measures how close our prediction is to the correct target.

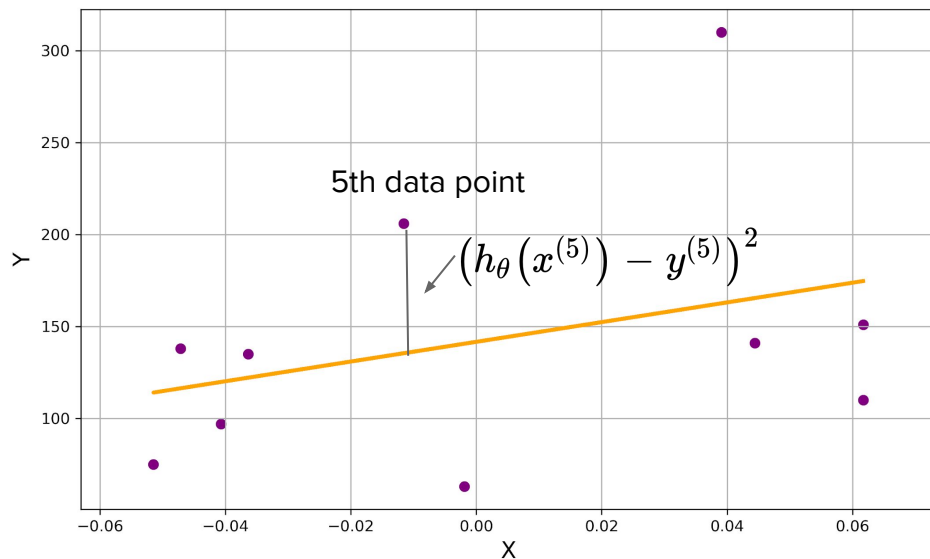


Square errors averaged over all training examples

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

Cost function - squared errors

A formula measures how close our prediction is to the correct target.



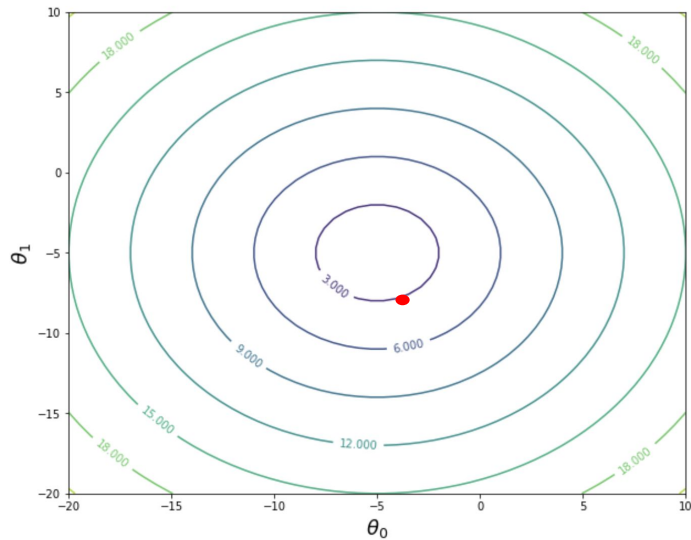
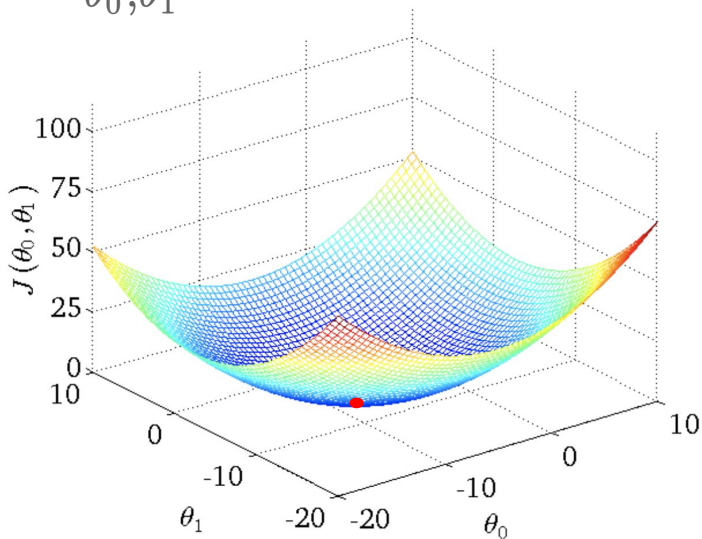
Square errors averaged over all training examples

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

The 1/2 factor is just to make the calculations convenient.

Surface and contour plot

Goal: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$



Optimization

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

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

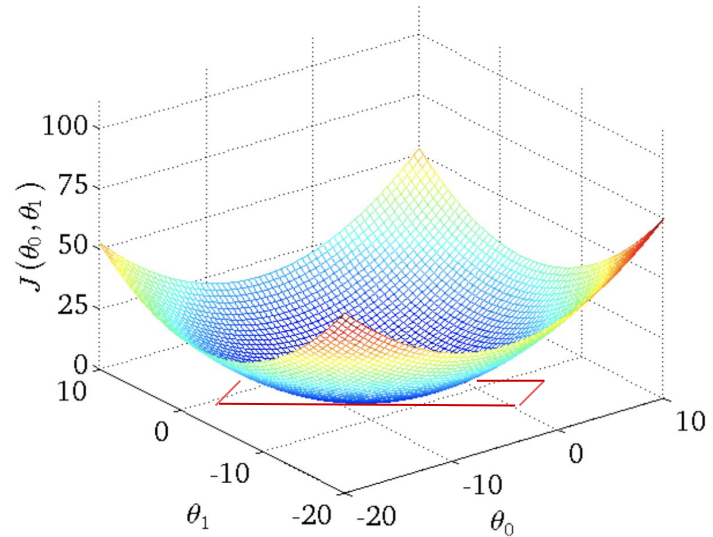
Goal: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

2 strategies for this optimization problem:

1. **Numerical/direct solution**
2. **Iterative method (e.g. gradient descent)**

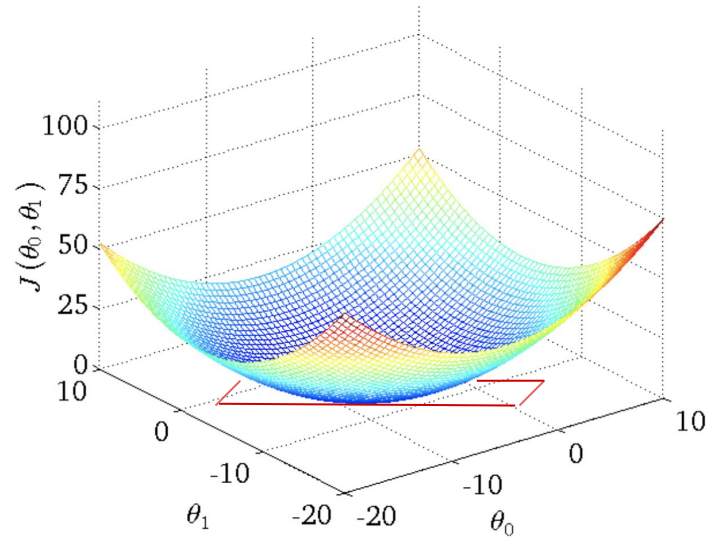
Direct solution

The global minimum in this mean square error function is achieved when all partial derivatives $\frac{\partial J}{\partial \theta_0} = 0$, $\frac{\partial J}{\partial \theta_1} = 0$ equal to 0. This works only in a handful of cases (e.g. linear regression).



Direct solution

The global minimum in this mean square error function is achieved when all partial derivatives $\frac{\partial J}{\partial \theta_0} = 0$, $\frac{\partial J}{\partial \theta_1} = 0$ equal to 0. This works only in a handful of cases (e.g. linear regression).



$$\begin{aligned}
\frac{\partial J}{\partial \theta_0} &= \frac{\partial}{\partial \theta_0} \left(\frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \right) \\
&= \frac{1}{2m} \frac{\partial}{\partial \theta_0} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2 \\
&= \frac{1}{2m} \frac{\partial}{\partial \theta_0} \sum_{i=1}^m (\theta_0^2 + (\theta_1 x^{(i)} - y^{(i)})^2 + 2(\theta_1 x^{(i)} - y^{(i)})\theta_0) \\
&= \frac{1}{2m} \left(2m\theta_0 + 2 \left(\sum_{i=1}^m \theta_1 x^{(i)} - y^{(i)} \right) \right) \\
&= \theta_0 + \frac{1}{m} \sum_{i=1}^m (\theta_1 x^{(i)} - y^{(i)}) \\
&= 0
\end{aligned}$$

$$\begin{aligned}
\frac{\partial J}{\partial \theta_1} &= \frac{\partial}{\partial \theta_1} \left(\frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \right) \\
&= \frac{1}{2m} \frac{\partial}{\partial \theta_1} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2 \\
&= \frac{1}{2m} \frac{\partial}{\partial \theta_1} \sum_{i=1}^m \left(\theta_1^2 (x^{(i)})^2 + (\theta_0 - y^{(i)})^2 + 2(\theta_0 - y^{(i)})\theta_1 x^{(i)} \right) \\
&= \frac{1}{m} \sum_{i=1}^m (x^{(i)})^2 \theta_1 + \frac{1}{m} \sum_{i=1}^m (\theta_0 - y^{(i)}) x^{(i)} \\
&= 0
\end{aligned}$$

Solve these first-order equations to obtain the optimal θ_0 and θ_1 .

$$\begin{cases} \theta_0 + \frac{1}{m} \sum_{i=1}^m (\theta_1 x^{(i)} - y^{(i)}) = 0 \\ \sum_{i=1}^m (x^{(i)})^2 \theta_1 + \sum_{i=1}^m (\theta_0 - y^{(i)}) x^{(i)} = 0 \end{cases}$$

Linear regression uses predictors matrix **X**

Size in feet^2 X1	# bedroom X2	# bathroom X3	# living room X4	House price in 10k Y
3081	4	3	2	110
2774	3	2	2	89
1645	3	3	1	63

$$X_{3 \times 4} = \begin{pmatrix} 3081 & 4 & 3 & 2 \\ 2774 & 3 & 2 & 2 \\ 1645 & 3 & 3 & 1 \end{pmatrix}$$

Hypothesis: $h_{\theta}(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

Linear regression uses predictors matrix **X**

Size in feet^2 X1	# bedroom X2	# bathroom X3	# living room X4	House price in 10k Y
3081	4	3	2	110
2774	3	2	2	89
1645	3	3	1	63

One feature across all training samples

$$X_{3 \times 4} = \begin{pmatrix} 3081 & 4 & 3 & 2 \\ 2774 & 3 & 2 & 2 \\ 1645 & 3 & 3 & 1 \end{pmatrix}$$

Hypothesis: $h_{\theta}(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

Linear regression uses predictors matrix **X**

Size in feet^2 X1	# bedroom X2	# bathroom X3	# living room X4	House price in 10k Y
3081	4	3	2	110
2774	3	2	2	89
1645	3	3	1	63

$$X_{3 \times 4} = \begin{pmatrix} 3081 & 4 & 3 & 2 \\ 2774 & 3 & 2 & 2 \\ 1645 & 3 & 3 & 1 \end{pmatrix}$$

One training example

Hypothesis: $h_{\theta}(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

Linear regression uses predictors matrix **X**

Size in feet^2 X1	# bedroom X2	# bathroom X3	# living room X4	House price in 10k Y
3081	4	3	2	110
2774	3	2	2	89
1645	3	3	1	63

$$X_{3 \times 5} = \begin{pmatrix} 1 & 3081 & 4 & 3 & 2 \\ 1 & 2774 & 3 & 2 & 2 \\ 1 & 1645 & 3 & 3 & 1 \end{pmatrix}$$

Hypothesis: $h_{\theta}(X) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

Let $\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{pmatrix}$ $Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$



$$Y = H_{\theta}(X) = X\theta$$

Direct solution for multivariate linear regression

$$H_{\theta}(X) = X\theta$$
$$\theta^* = (X^T X)^{-1} X^T y$$

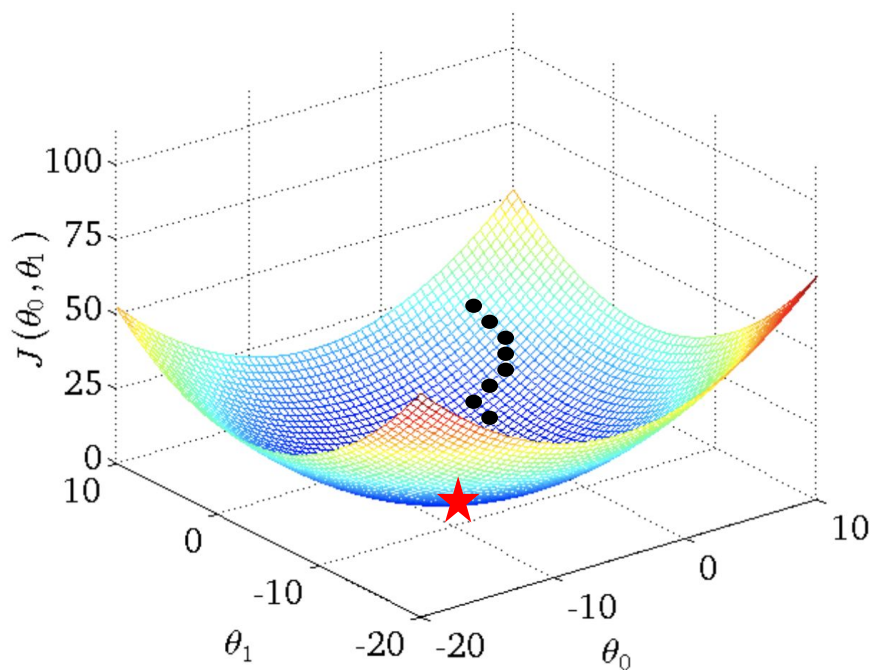
What if $X^T X$ is non-invertible?

- Use `pinv()` to compute $(X^T X)^{-1}$, rather than `inv()`.

When $X^T X$ is non-invertible?

- Redundant features (linearly dependent)
 - $x_1 = \text{size in feet}^2$
 - $x_2 = \text{size in meter}^2$
- Too many features (sample size < features)
 - Delete some features
 - Use regularization

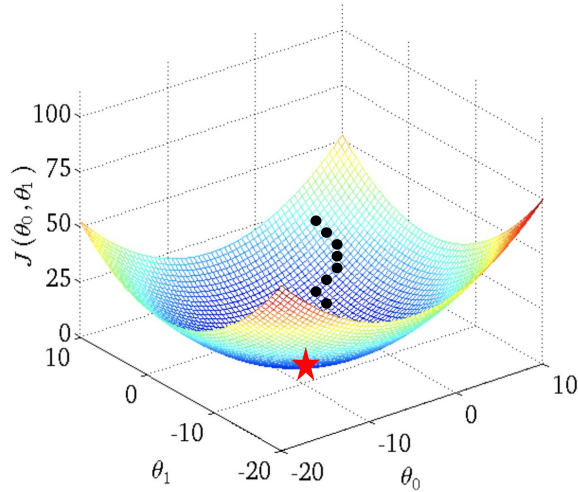
Iterative method



Mean square error cost:

- Convex function
- Bowl shape

Iterative method



1. [Initialization] start off somewhere reasonable on the error surface.
2. Evaluate our current loss.
3. Figure out a direction to move towards a position on the error surface which have a lower loss.
4. [Iterate] repeat this process until converge [criterion].

Gradient Descent

Gradient descent algorithm

Repeat until convergence:

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

Gradient descent algorithm

Repeat until convergence:

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

Learning rate

Gradient descent algorithm

Repeat until convergence:

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

Partial derivative

Gradient descent algorithm

Repeat until convergence:

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

For $j=0, 1$, **correct simultaneous update**:

$$temp_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta)$$

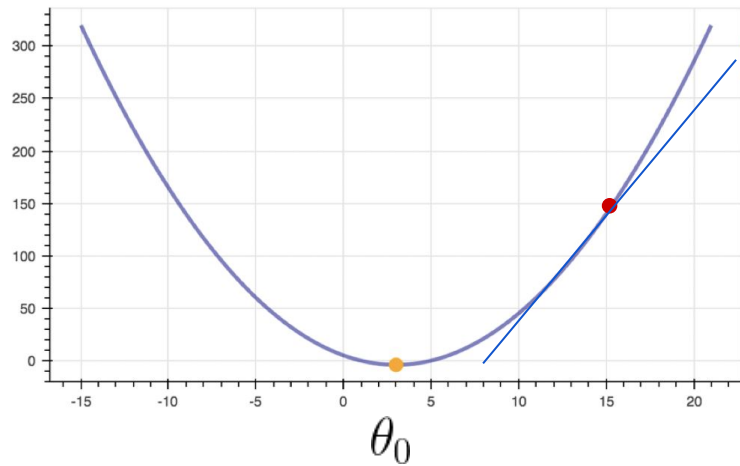
$$temp_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta)$$

$$\theta_0 = temp_0$$

$$\theta_1 = temp_1$$

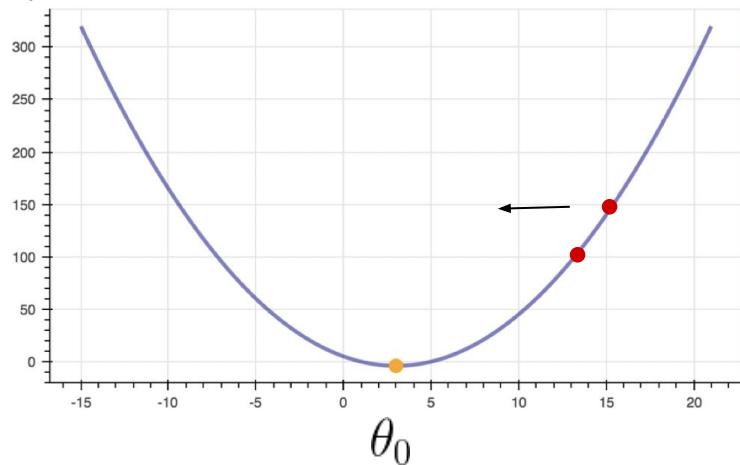
Gradient descent

$$J(\theta_0)$$



Gradient descent

$J(\theta_0)$



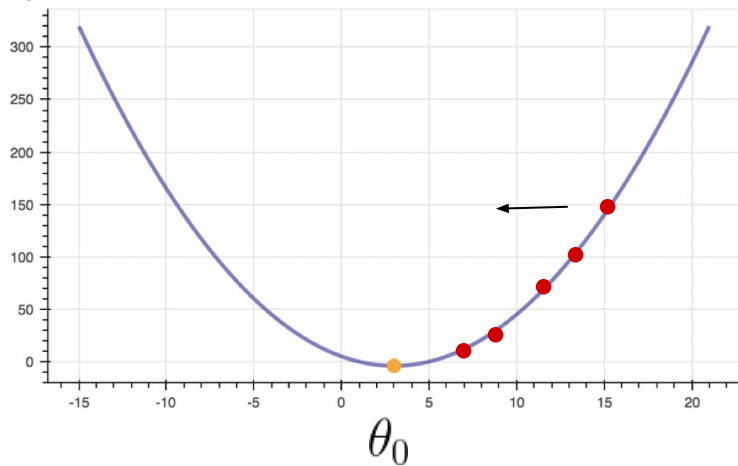
This tangent line has
positive slope

$$\frac{d}{d\theta_0} J(\theta_0) > 0$$

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

Gradient descent

$J(\theta_0)$



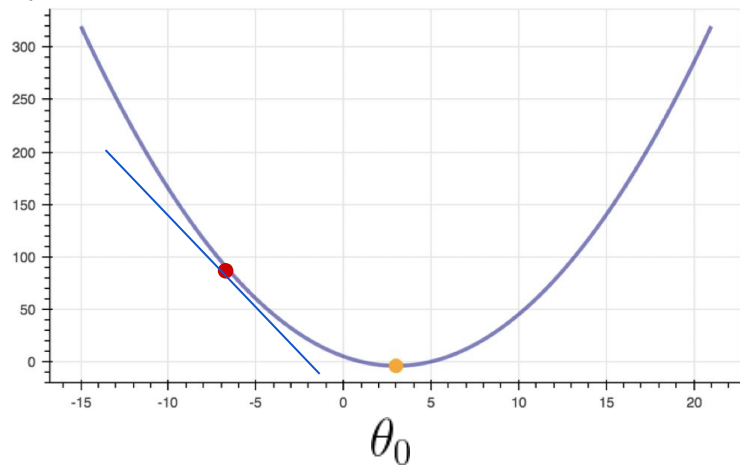
This tangent line has
positive slope

$$\frac{d}{d\theta_0} J(\theta_0) > 0$$

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

Gradient descent

$J(\theta_0)$



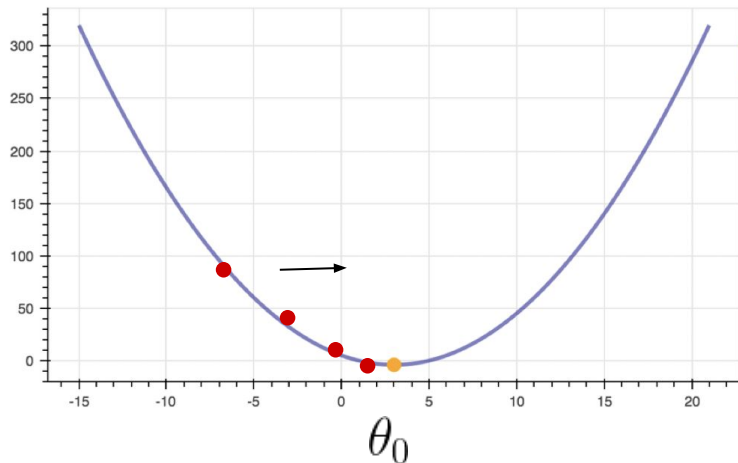
This tangent line has
negative slope

$$\frac{d}{d\theta_0} J(\theta_0) < 0$$

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

Gradient descent

$J(\theta_0)$



This tangent line has negative slope

$$\frac{d}{d\theta_0} J(\theta_0) < 0$$

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

- Gradient points to the direction of the greatest rate of increase of the function.
- The negative gradient is the direction of the **fastest decrease**.
- Therefore, if we update the parameters by moving towards the direction of negative gradient (thus “gradient descent”), the loss is **expected** to decrease.

Gradient descent for linear regression

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

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

Repeat until convergence {

Batch gradient descent

$$\begin{aligned}\theta_0 &= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \\ \theta_1 &= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}\end{aligned}$$

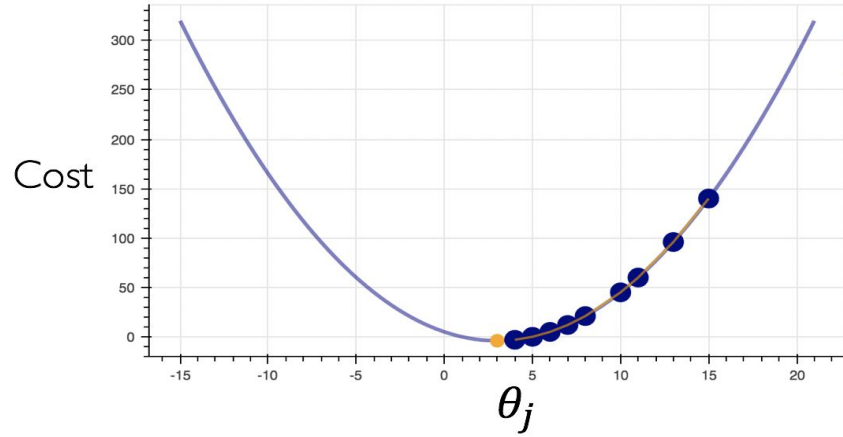
}

Batch VS stochastic gradient descent

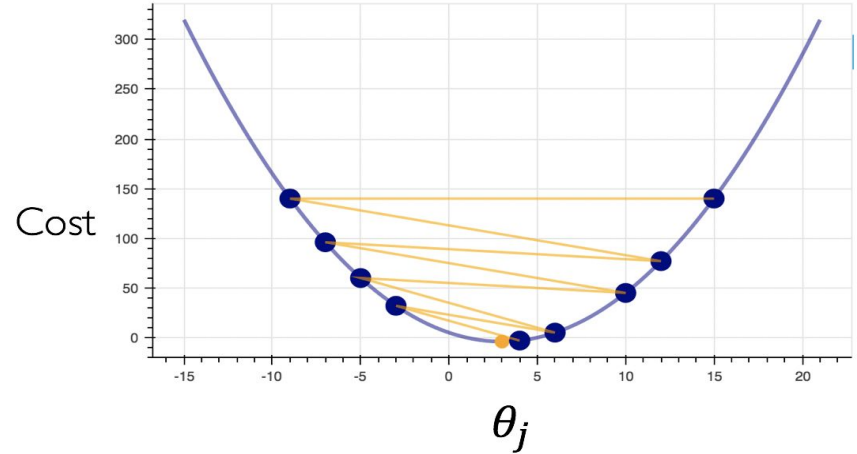
In both batch gradient descent and stochastic gradient descent (SGD), you update a set of parameters in an iterative manner to minimize an error function.

- **Batch GD:** you have to run through ALL the samples in your training set to do a single update for a parameter in a particular iteration.
- **SGD:** on the other hand, you use ONLY ONE or SUBSET of training sample.

Learning rate

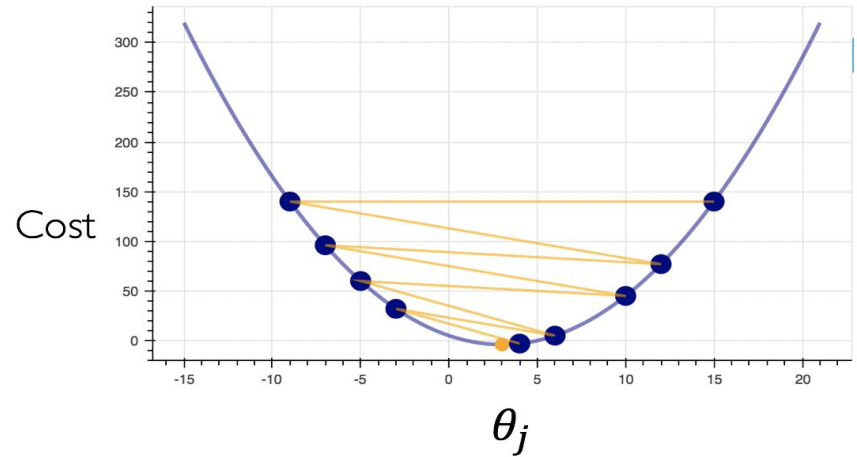
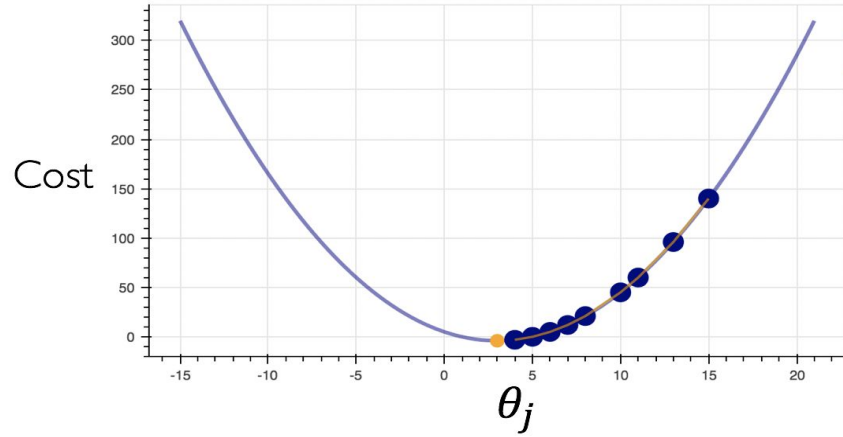


If learning rate is too small, gradient descent can be very slow to converge.

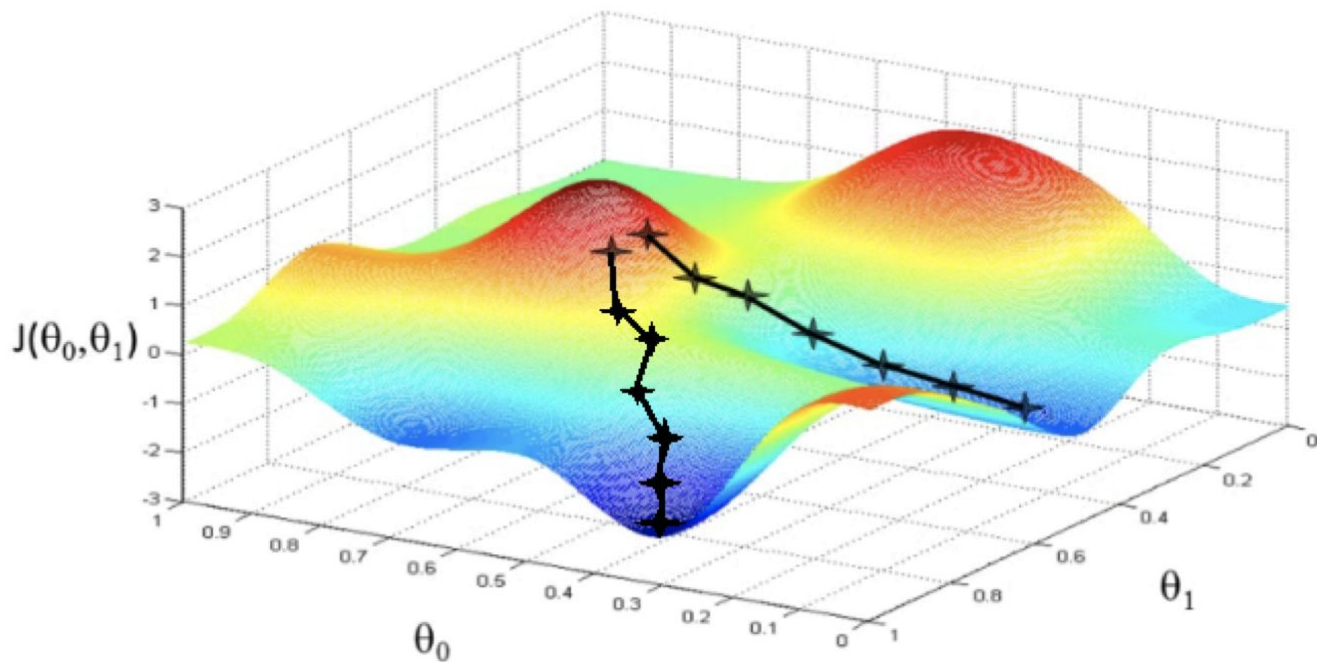


If learning rate is too big, gradient descent can overshoot the minimum. It may even fail to converge, or even diverge.

Learning rate



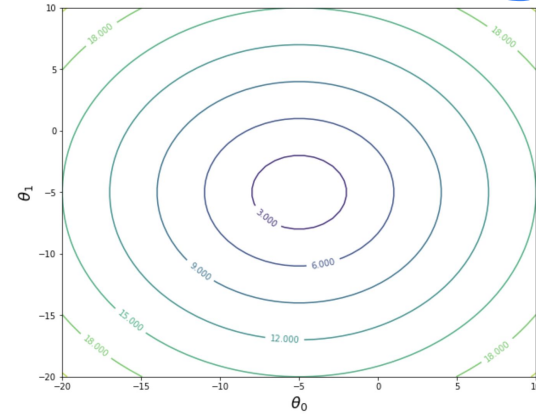
- Gradient descent can converge to a local minimum, even with the learning rate fixed.
- We will see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001.



Depends on the way you initialize the parameters, we may end with different local optimals.

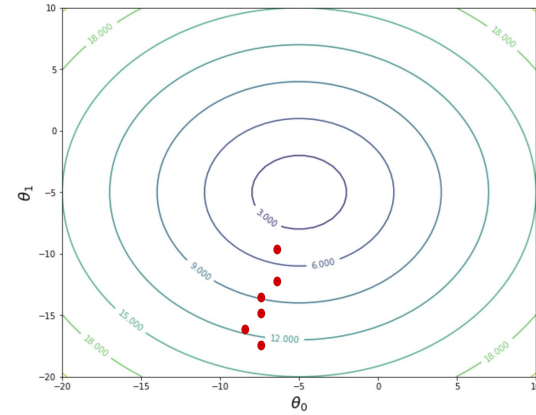
Gradient in practice - feature scaling

# bedroom X1	# bathroom X2	House price in 10k Y
4	3	110
3	2	89
3	3	63



Gradient in practice - feature scaling

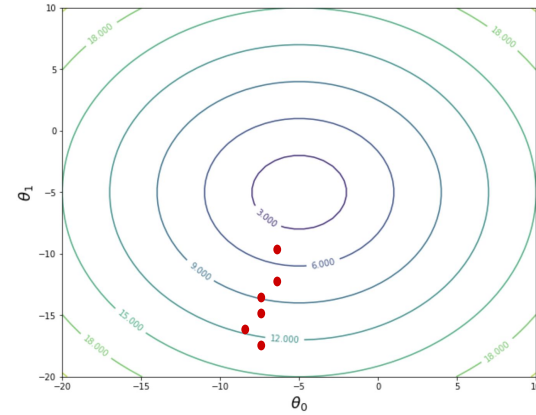
# bedroom X1	# bathroom X2	House price in 10k Y
4	3	110
3	2	89
3	3	63



Gradient in practice - feature scaling

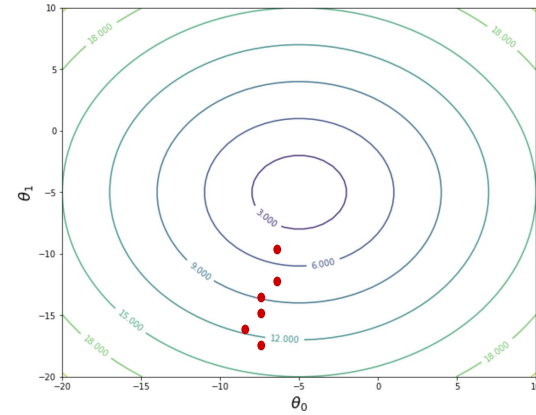
# bedroom X1	# bathroom X2	House price in 10k Y
4	3	110
3	2	89
3	3	63

Size in feet^2 X1	# bathroom X2	House price in 10k Y
3081	4	110
2774	3	89
1645	3	63

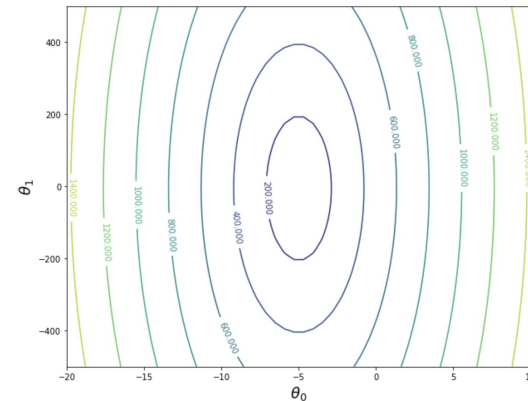


Gradient in practice - feature scaling

# bedroom X1	# bathroom X2	House price in 10k Y
4	3	110
3	2	89
3	3	63

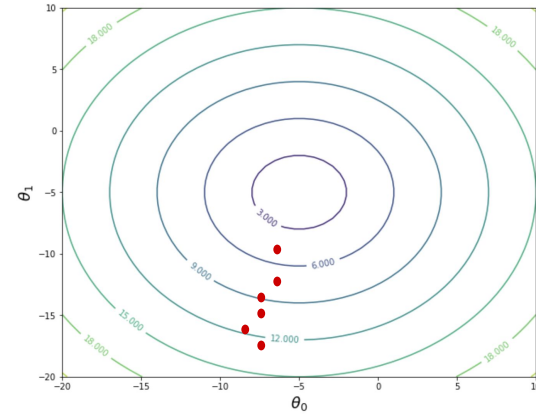


Size in feet^2 X1	# bathroom X2	House price in 10k Y
3081	4	110
2774	3	89
1645	3	63

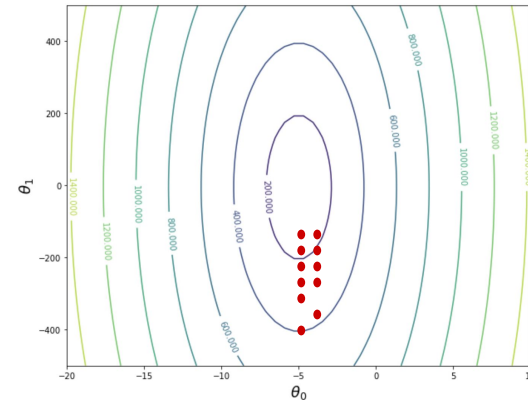


Gradient in practice - feature scaling

# bedroom X1	# bathroom X2	House price in 10k Y
4	3	110
3	2	89
3	3	63



Size in feet^2 X1	# bathroom X2	House price in 10k Y
3081	4	110
2774	3	89
1645	3	63



Feature scaling

Size in feet^2 X1	# bathroom X2	House price in 10k Y
3081	4	110
2774	3	89
1645	3	63

$$x_{rescaled} = \frac{x_i^{(i)} - \min(x_i)}{\max(x_i) - \min(x_i)}$$

Min-max scaler

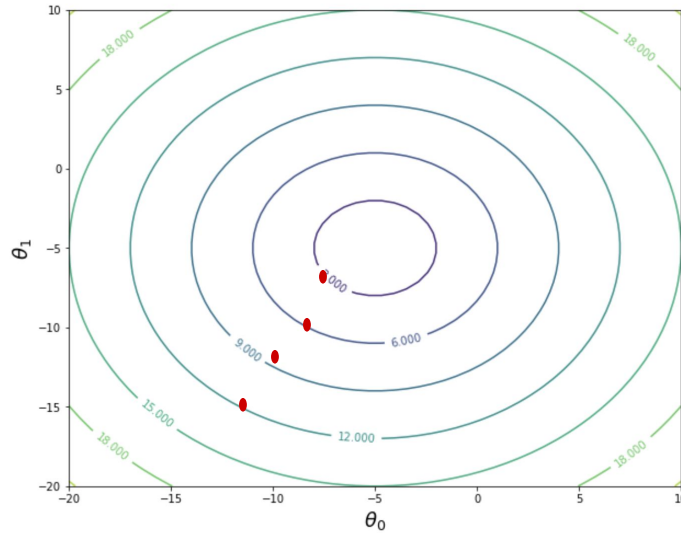
Standard scaler

$$x_{rescaled} = \frac{x_i^{(i)} - \text{mean}(x_i)}{\text{std}(x_i)}$$

Size in feet^2 X1-rescale	# bathroom X2-rescale	House price in 10k Y
1.0	1	110
0.79	0	89
0.0	0	63

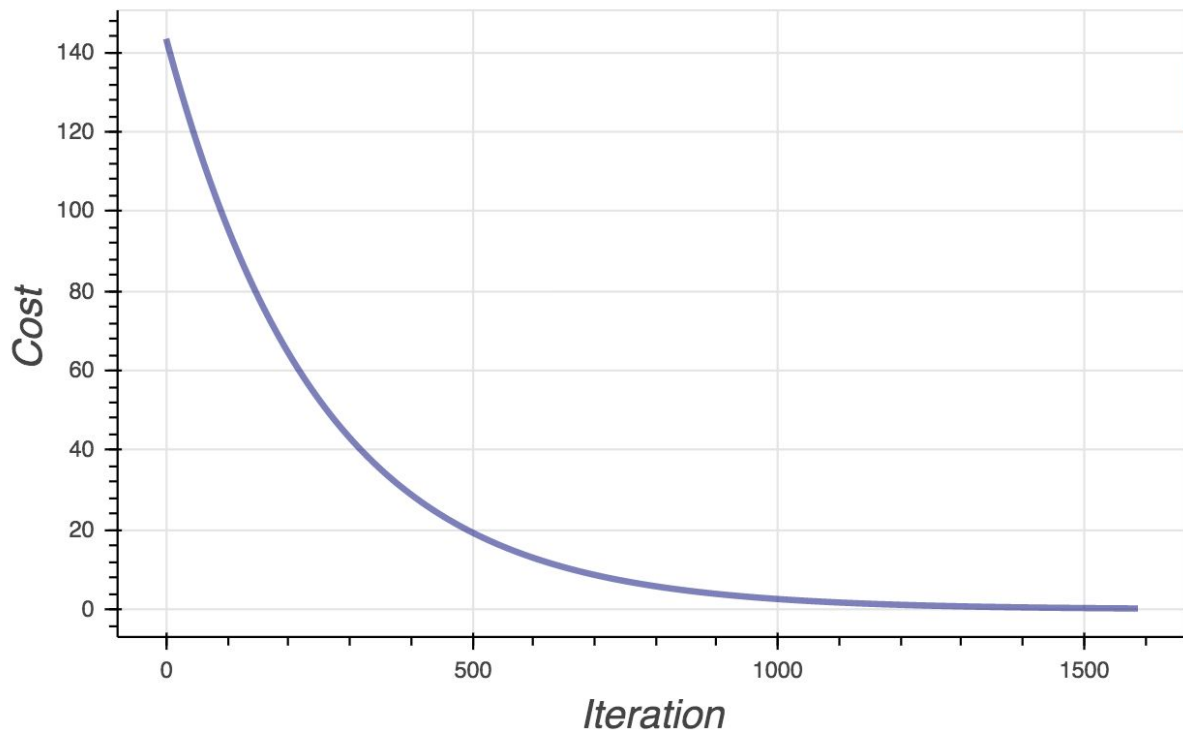
Size in feet^2 X1-rescale	# bathroom X2-rescale	House price in 10k Y
0.94	1.41	110
0.44	-0.71	89
-1.38	-0.71	63

Feature scaling



- Gradient descent is very sensitive to the scale of features.
- But direct solution $\theta^* = (X^T X)^{-1} X^T y$ isn't.

Make sure gradient descent is working



Declare convergence

If cost function J decrease by less than 0.0001 in one iteration.

Gradient descent VS direct solution

When we have m samples, n features.

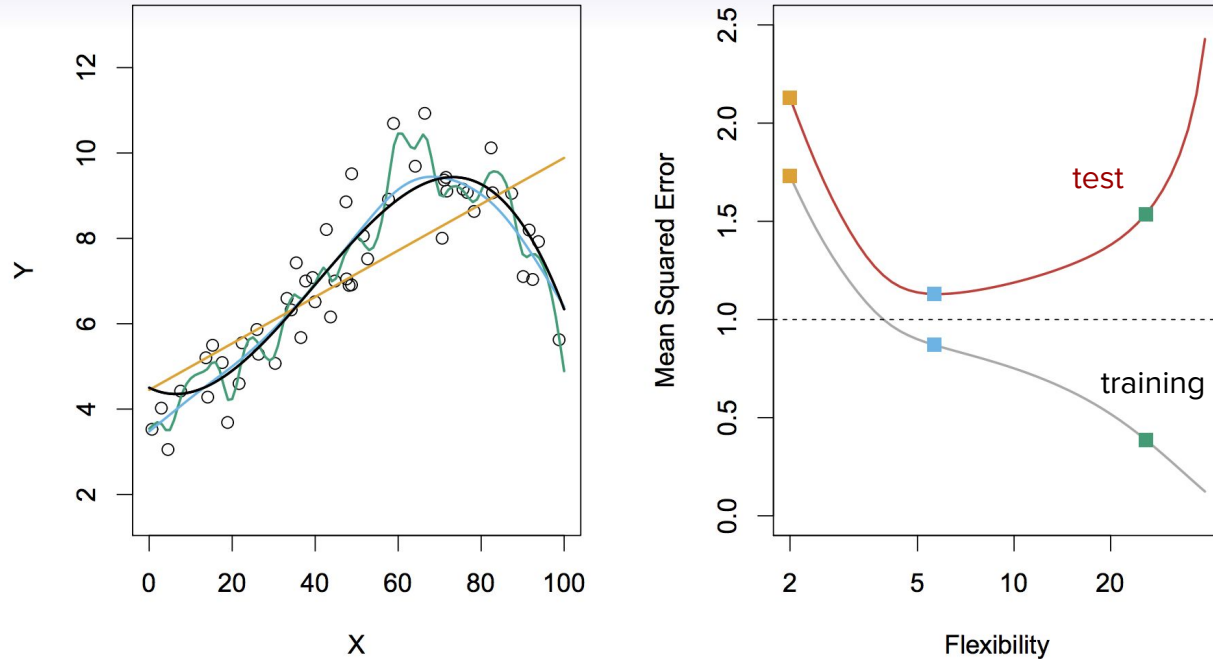
Gradient descent

- Need to choose learning rate: α
- Need many iterations
- Work well even when n is large
- Can be applied to a much broader set of models.

Direct solution

- Need to compute $(X^T X)^{-1}$
- Slow if n is large

Generalization



Black curve is truth. Red curve on right is MSE from test data, grey curve is MSE from training data. Orange, blue and green curves/squares correspond to fits of different flexibility.

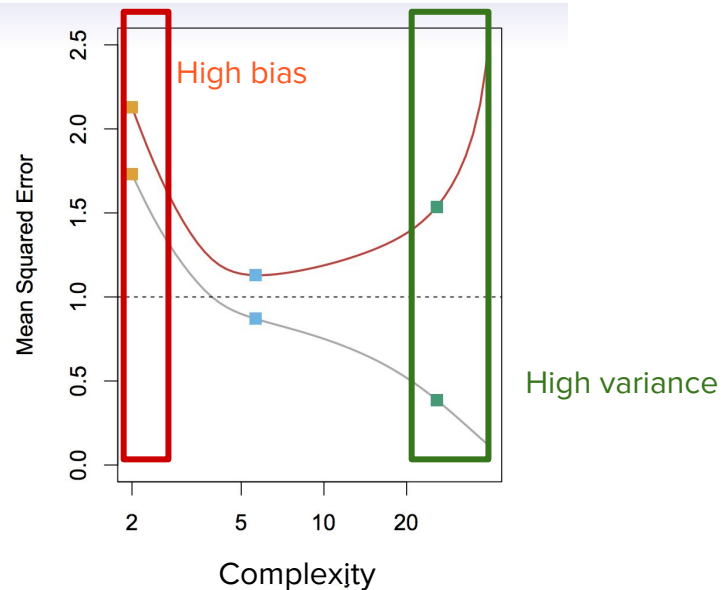
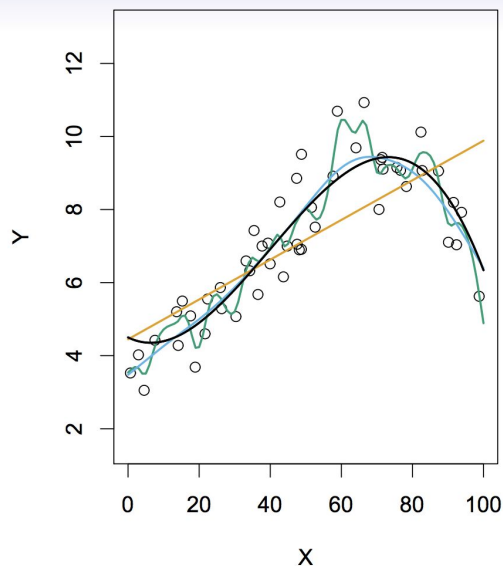
Generalization

- **Underfitting**: the model is too simple - does not fit the data.
- **Overfitting**: the model is too complex - fits perfectly, does not generalize.
- We would like our models to generalize to data they haven't seen before.

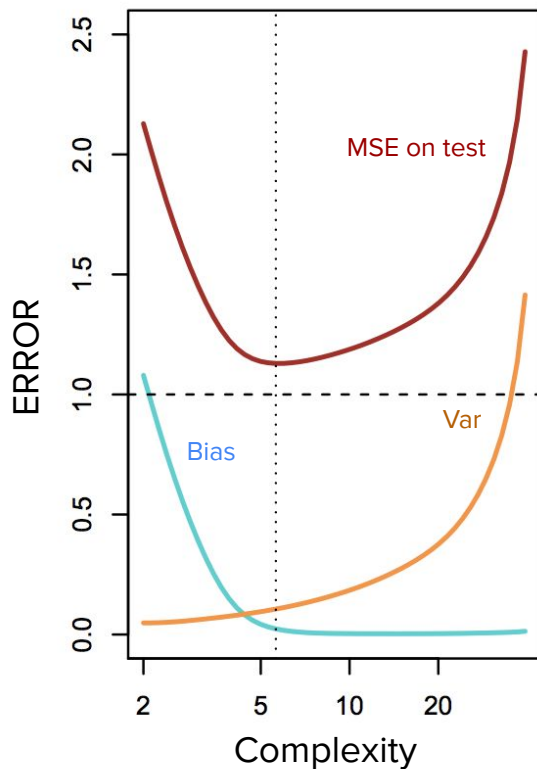
Tune hyperparameters (e.g. the degree of the polynomial) using a validation set, or select subset of features.

Bias-variance trade-off

- Variance: the difference in fits between data sets.
- Bias: the inability for a algorithm to capture the true relationship



Bias-variance trade-off



$$E(y_0 - \hat{y}(x_0))^2 \approx \text{Var}(\hat{y}(x_0)) + [\text{Bias}(\hat{y}(x_0))]^2$$

Typically as the complexity of the model increases, its variance increases, and its bias decreases. **So choosing the complexity based on average test error amounts to a bias-variance trade-off.**

Addressing overfitting

Options:

1. Manually select which features to stay
2. Model selection method (forward/backward stepwise)
3. Regularization:
 - a. Keep all the features, but constrain the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.
 - b. Drop out some training data

Regularization - L2

In linear regression, the goal is to:

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

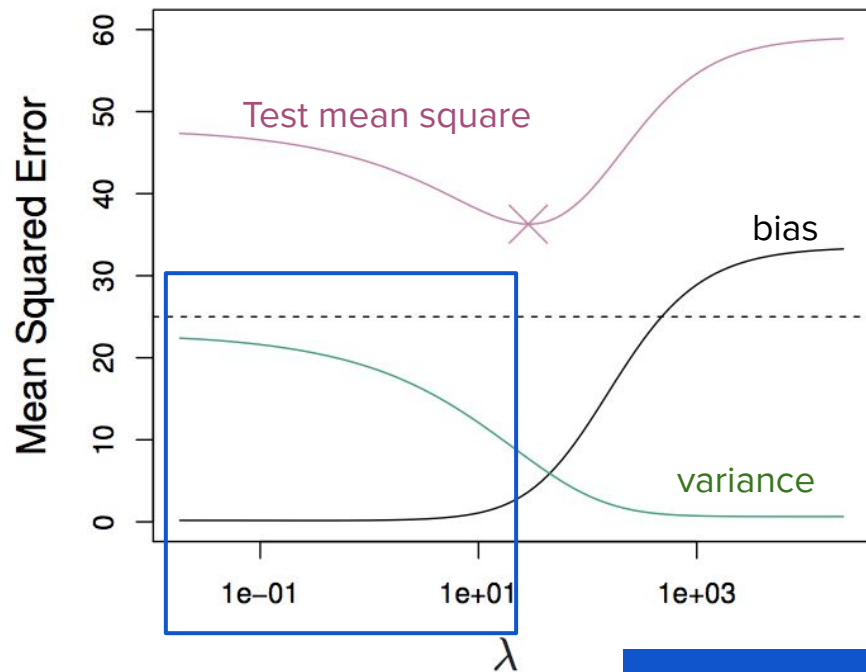
After adding L2 regularization element, the goal has changed to:

Ridge regression

$$\min J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2$$

L2 penalty term

Improvement by using regularization



Simulated data with 50 samples, 45 features, all having nonzero coefficients.

As we make lambda larger, it almost makes no change in the bias, but variance drops.

The ridge regression get rewarded for the MSE goes down at $1e+01$.

Lambda needs to be chosen carefully!!!

Regularization - L1

L2 regularization:

Ridge regression

$$\min J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

L2 penalty term

L1 regularization:

Lasso regression

$$\min J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j| \right]$$

L1 penalty term

Regularization - L1

The L1 penalty has the effect of forcing some of the coefficient estimates to be **exactly equal to zero** when the regularization parameter λ is sufficiently large.

L1 regularization:

Lasso regression

$$\min J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j|$$

L1 penalty term

Gradient descent with L2 regularization

Repeat until convergence:

$$\theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j = \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$0 < \text{value} < 1$, shrinkage effect

Direct solution with L2 regularization

Direct solution:

$$H_{\theta}(X) = X\theta$$
$$\theta^* = (X^T X)^{-1} X^T y$$

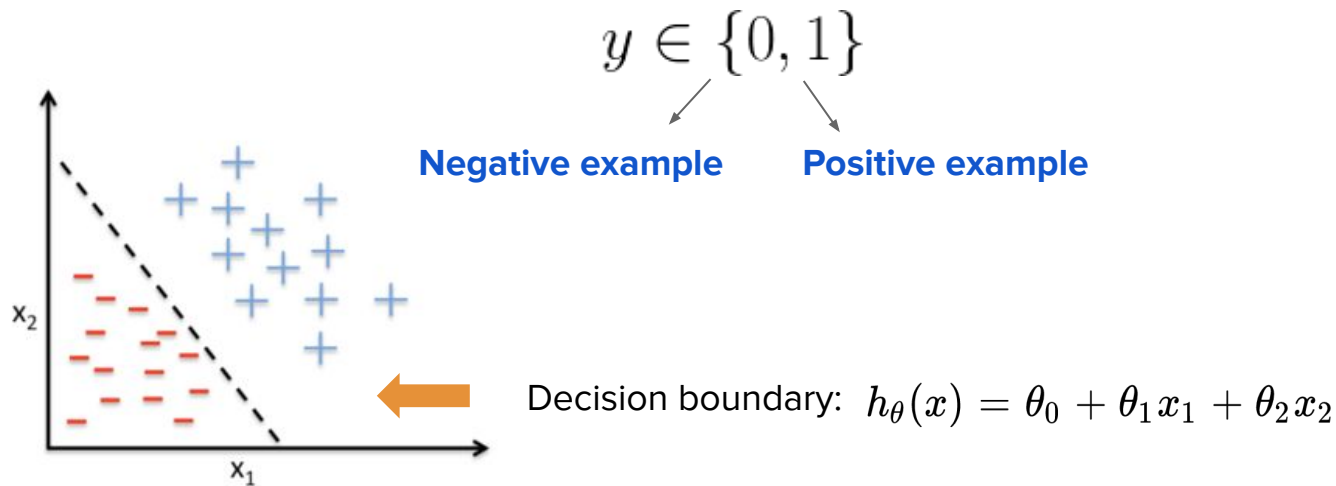
With regularization:

$$\theta^* = \left(X^T X + \lambda \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} \right)^{-1} X^T y$$

Classification

Binary linear classification

- **Classification:** predict a discrete-valued target.
- **Binary classification:** predict a binary target
 - Example: survivals, email spams, frauds



Logistic regression

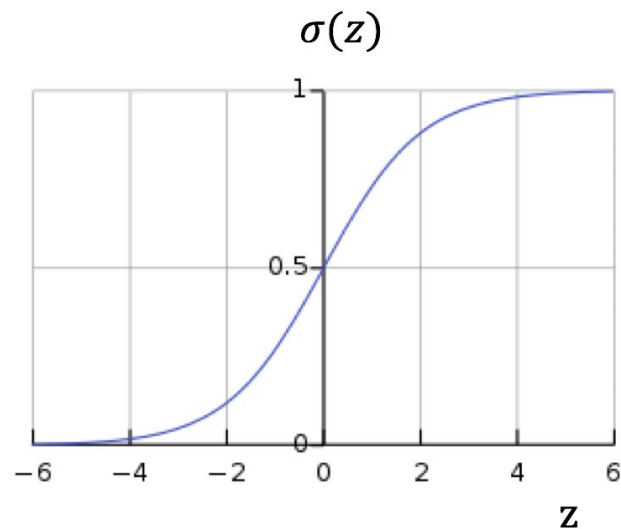
Hypothesis:

$$h_{\theta}(x) = g(\theta^T x)$$

Activation function

$$g(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid/logistic
function



$$\hat{g}(\theta^T x) \in (0, 1)$$

This function can model output probabilities naturally.

$$\hat{g}(\theta^T x) = P(y = 1 | x, \theta)$$

Example: given x , and all parameters, the predicted value can be interpreted as the probability that true target equals to 1.

$$h_{\theta}(x = 1, \theta_0 = 3, \theta_1 = 2) = \frac{1}{1 + e^{-(3+2*1)}} = 0.993$$

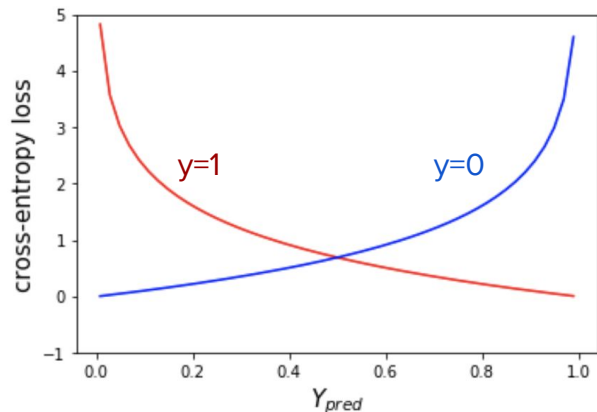
Logistic regression

Hypothesis:
$$h_{\theta}(x) = \frac{1}{1 + e^{-(\theta^T x)}}$$

binary cross-entropy (or log-loss)

Cost:
$$\text{cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & y = 1 \\ -\log(1 - h_{\theta}(x)) & y = 0 \end{cases}$$

$$= -y \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x))$$



Logistic regression

$$\begin{aligned} \text{cost}(h_{\theta}(x), y) &= \begin{cases} -\log(h_{\theta}(x)) & y = 1 \\ -\log(1 - h_{\theta}(x)) & y = 0 \end{cases} \\ &= -y\log(h_{\theta}(x)) - (1 - y)\log(1 - h_{\theta}(x)) \end{aligned}$$

The cost function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{cost}(h_{\theta}(x^{(i)}) - y^{(i)})$$

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]$$

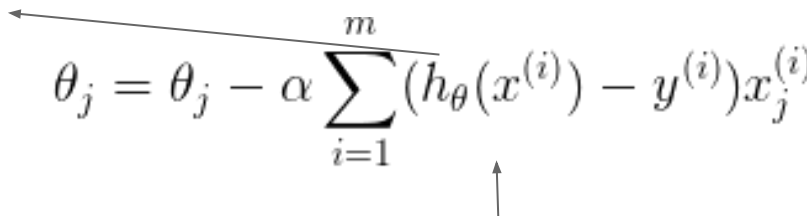
Gradient descent

Goal: minimize $J(\theta)$

Repeat until convergence:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \text{correct simultaneous update}$$

$$h_{\theta}(x) = \frac{1}{1 + e^{-(\theta^T x)}}$$

$$\theta_j = \theta_j - \alpha \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$


Looks identical to the linear regression!!!