Supervised Machine Learning: Regression and Classfication

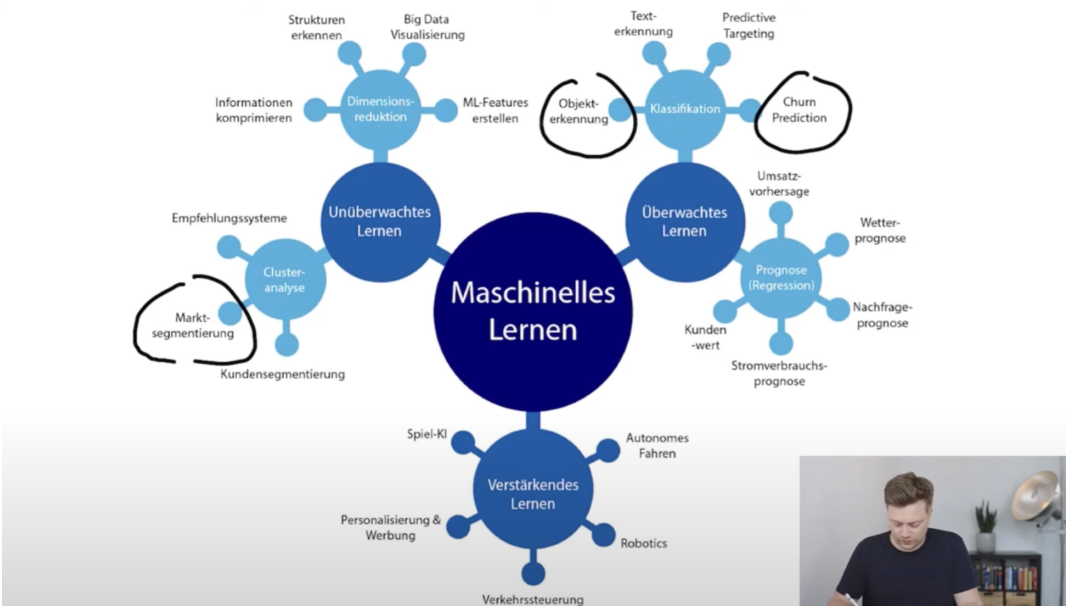
Course name: Supervised Machine Learning: Regression and Classification

Institution: Deeplearning.ai

By: Vadhna Samedy Hun

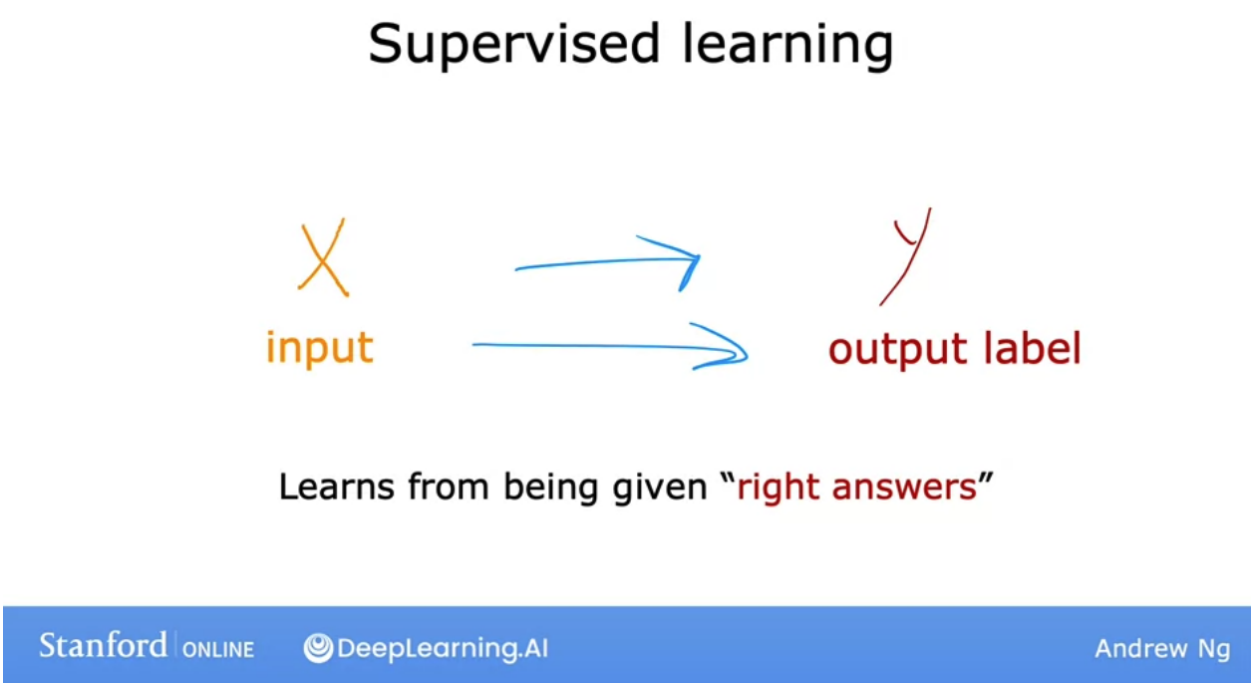
I. Introduction (Grundlage)

Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed. (Maschinelles Lernen is das Studienfach, das Computern die Fähigkeit gibt, zu lernen, ohne explizit programmiert zu werden.)



II. Machine Learning

2.1. Supervised Learning



Input (X)	Output (Y)	Application
email	spam? (0/1)	spam filtering
audio	text transcripts	speech recognition
English	Spanish	machine translation
ad, user info	click? (0/1)	online advertising
image, radar info	position of other cars	self-driving car
image of phone	defect? (0/1)	visual inspection

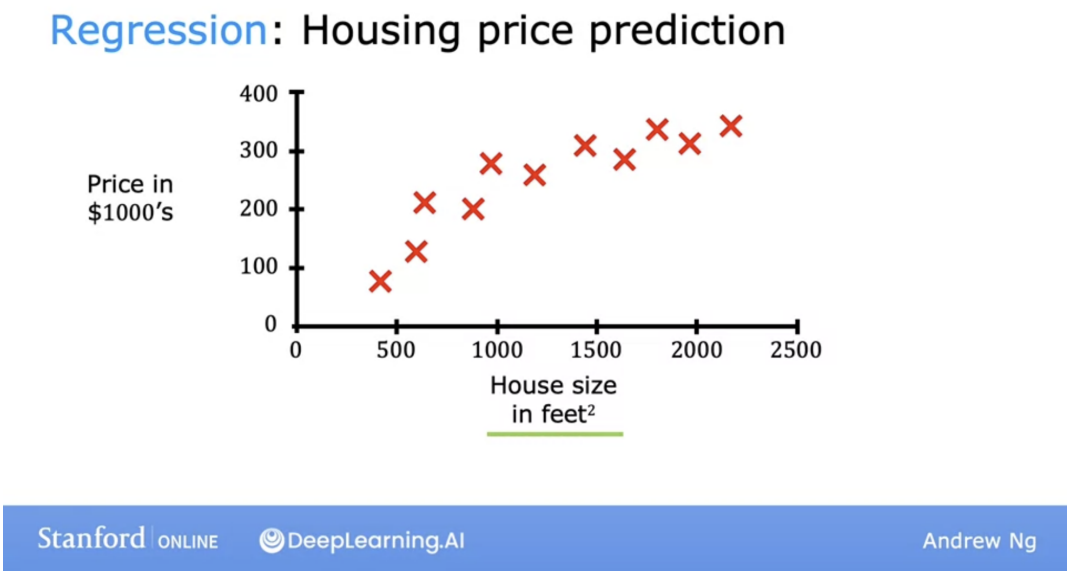
Stanford ONLINE DeepLearning.AI Andrew Ng

- There are 2 types of supervised learning:

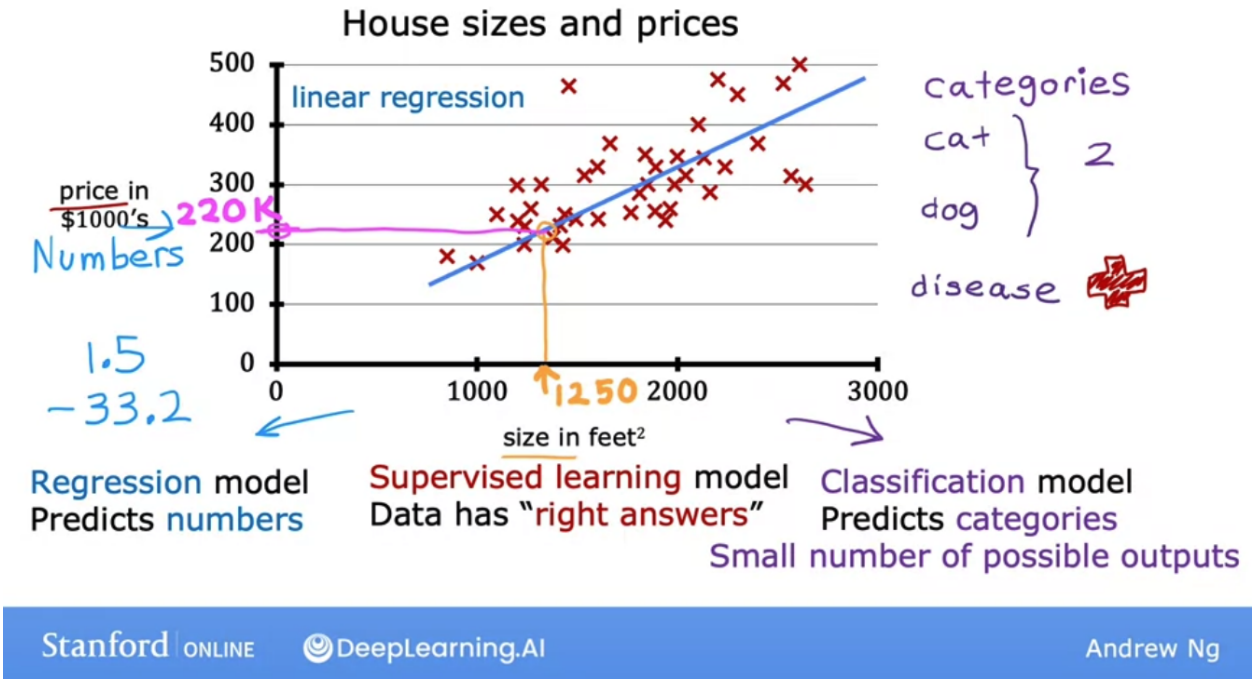
  Regression: predict continuous value

  Classification (Logistic Regression): predict categories (two categories)

Learn from data labeled with the “right answers”.



2.1.1. Regression



Terminology

Training set: Data used to train the model

	x size in feet^2	y price in \$1000's
(1)	2104	400
(2)	1416	232
(3)	1534	315
(4)	852	178
...	...	...
(47)	3210	870

$m = 47$

$x = 2104$ $y = 400$

$(x, y) = (2104, 400)$

Notation:

x = "input" variable
feature

y = "output" variable
"target" variable

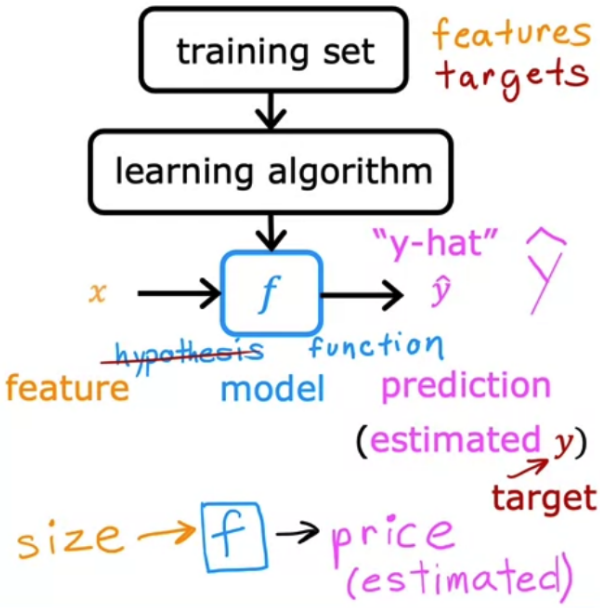
m = number of training examples

(x, y) = single training example

$(x^{(i)}, y^{(i)})$

$(x^{(i)}, y^{(i)})$ = i^{th} training example
(1^{st}, 2^{nd}, 3^{rd} ...)

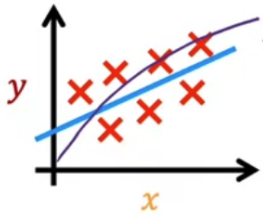
Stanford ONLINE DeepLearning.AI Andrew Ng



How to represent f?

$f_{w,b}(x) = wx + b$

$f(x)$

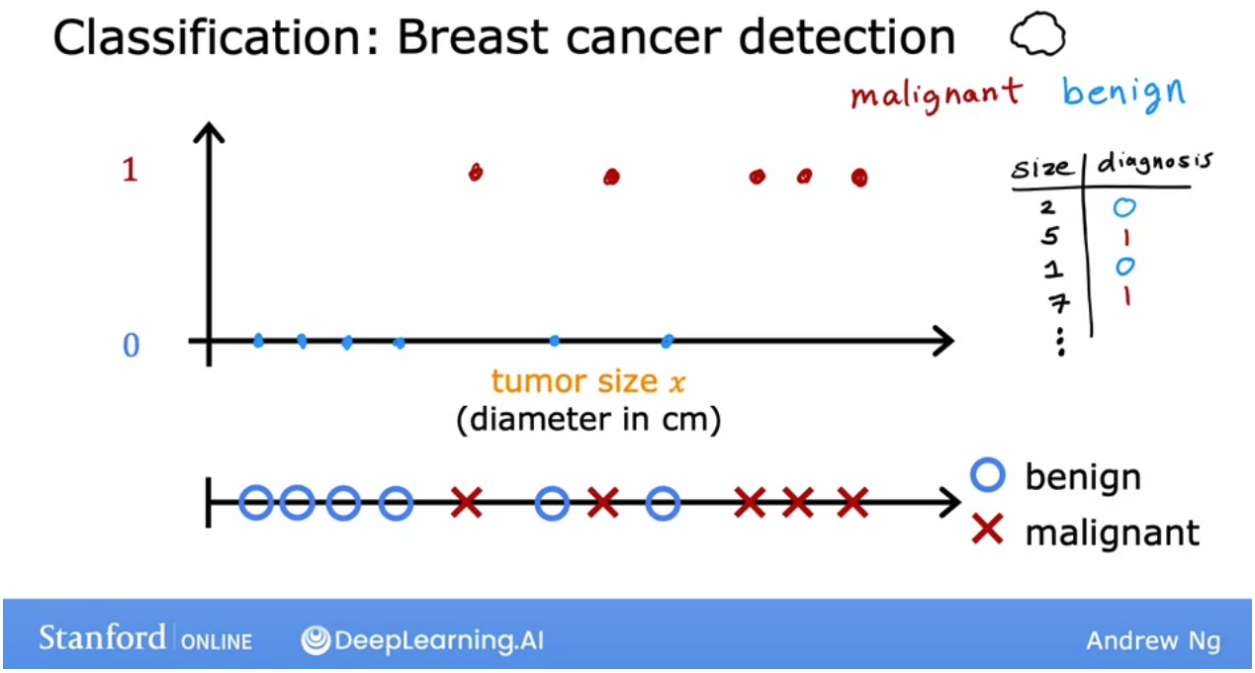


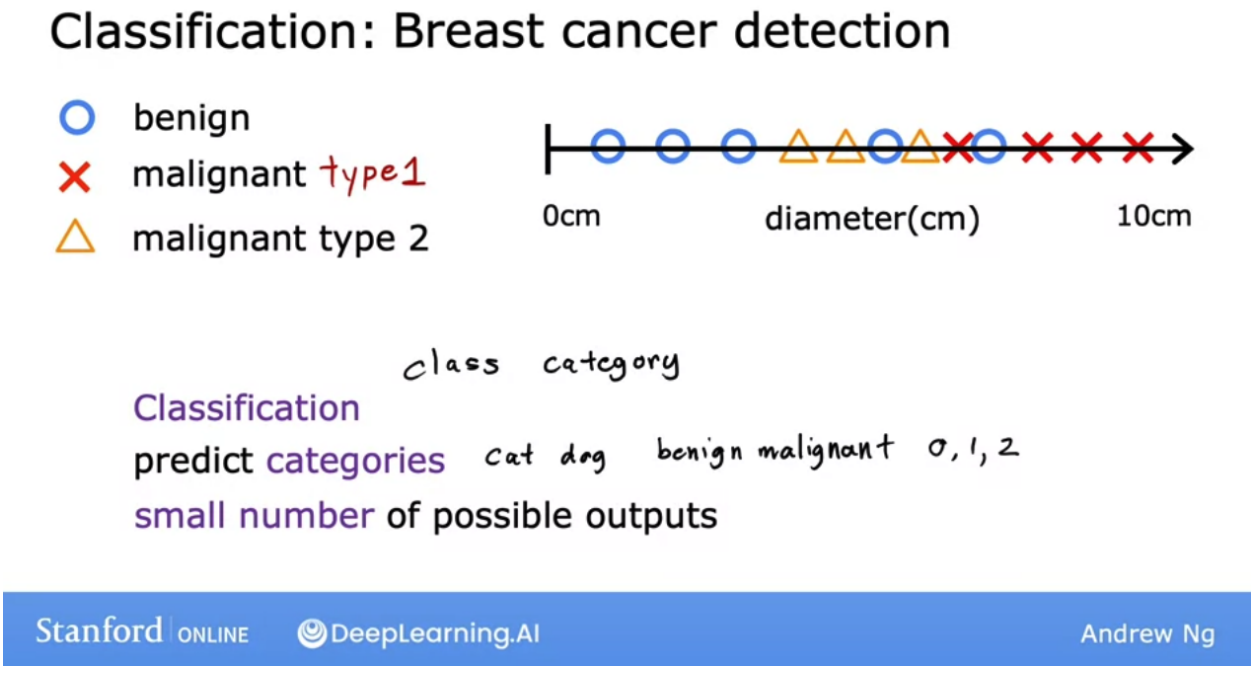
Linear regression with one variable.
size

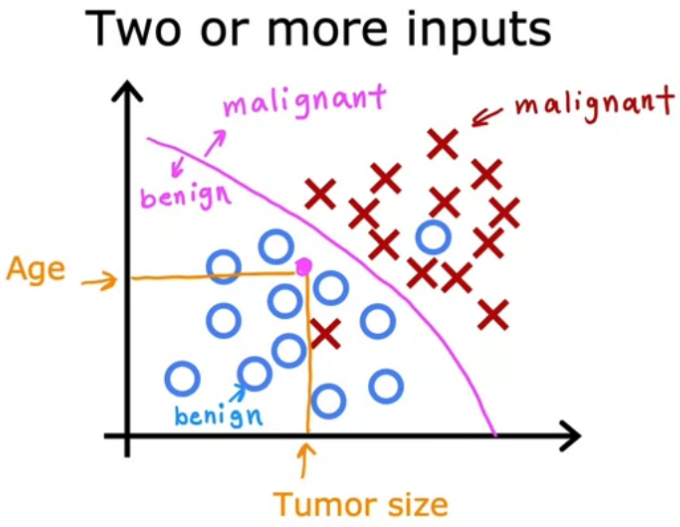
Univariate linear regression.
one variable

Stanford ONLINE DeepLearning.AI Andrew Ng

2.1.2. Classification

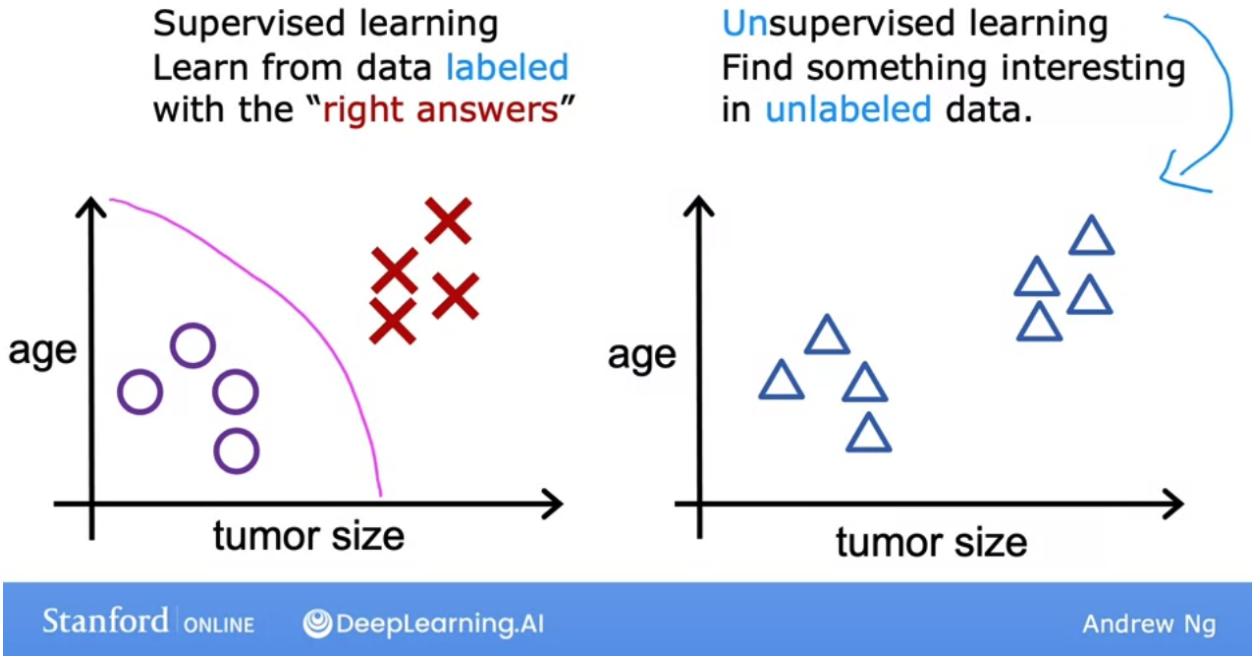


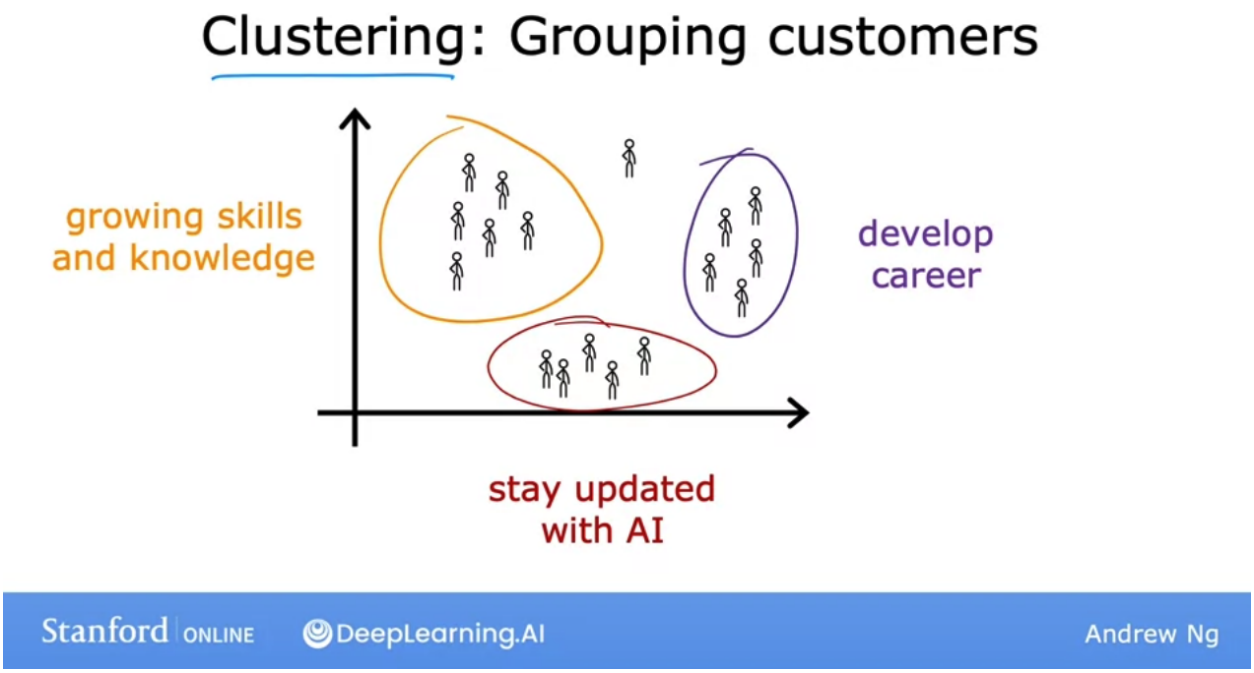




2.2. Unsupervised Learning

Find something interesting in unlabeled data.





Data only comes with inputs x, but not output labels y. Algorithm has to find structure in the data.

▼ Clustering

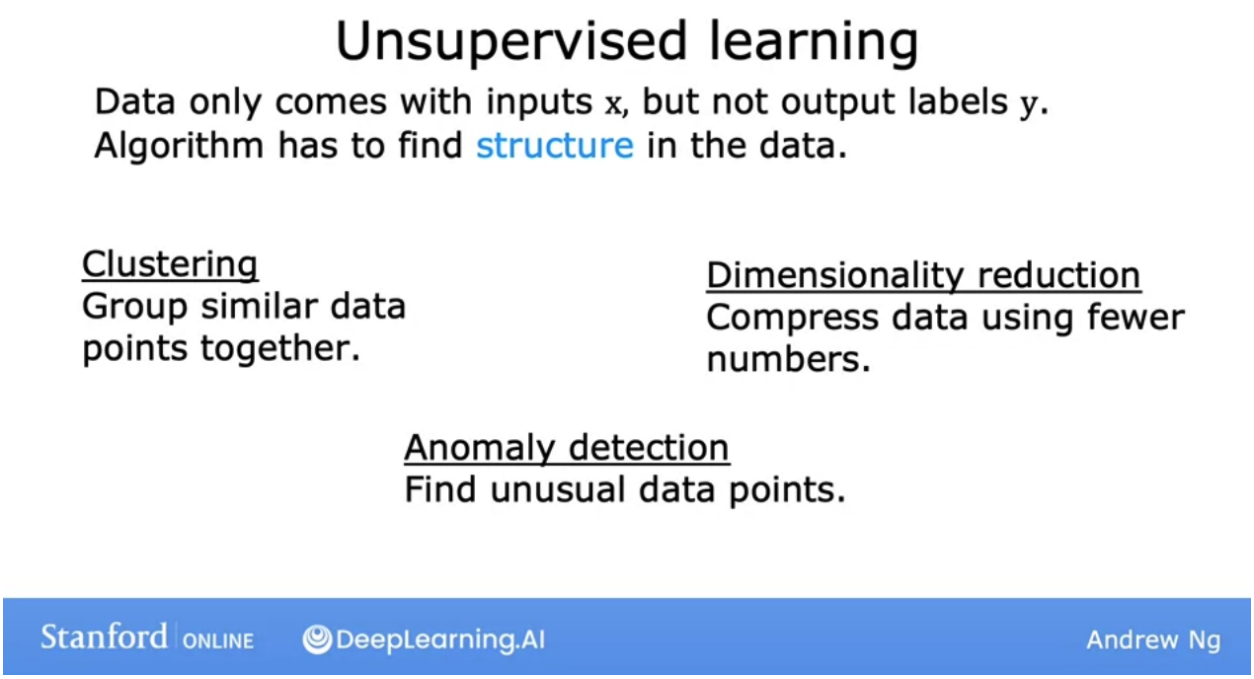
Group similar data points together.

▼ Anomaly detection

Find unusual data points.

▼ Dimensionality reduction

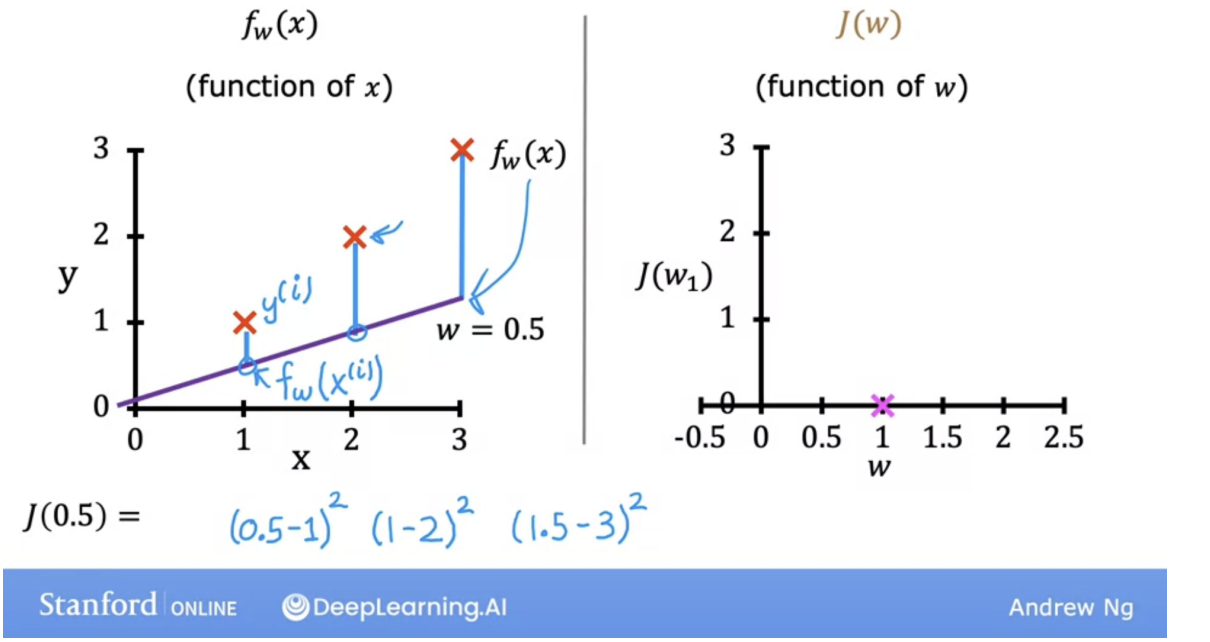
Compress data using fewer numbers.

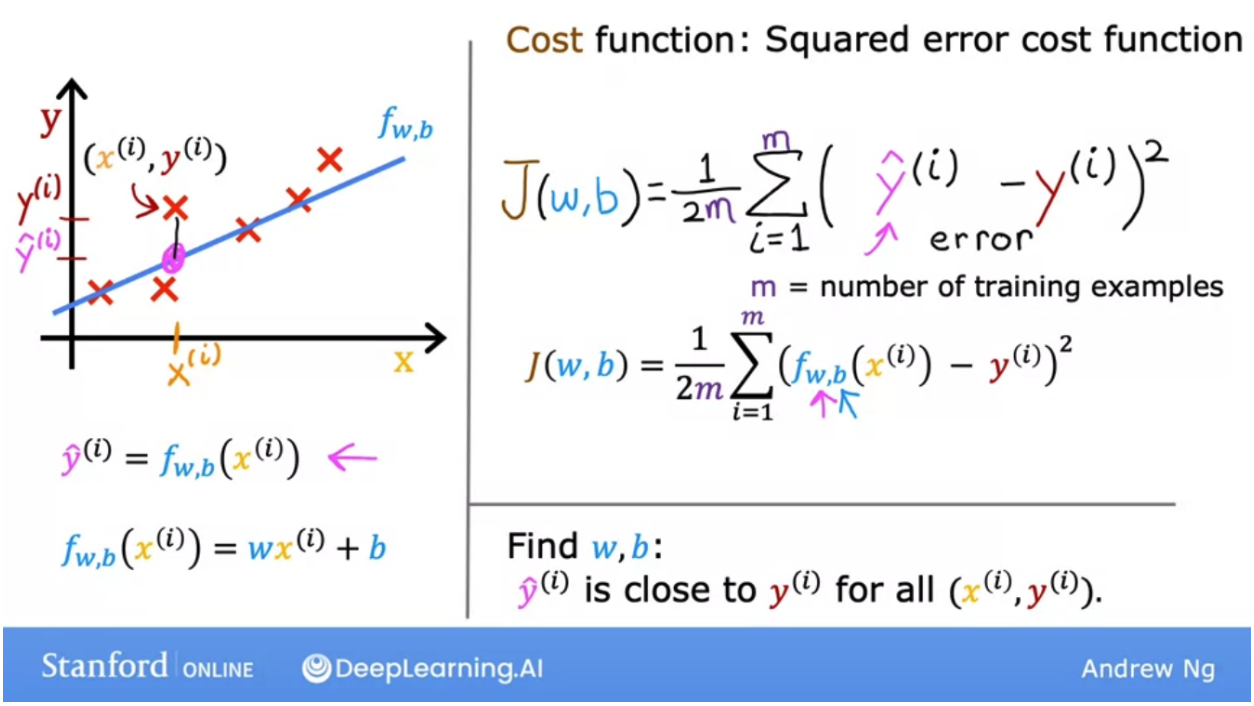


SupervisedMachineLearning - WEEK 1 - QUIZ 1

2.3. Cost function

Cost function is telling us how well our model is doing.



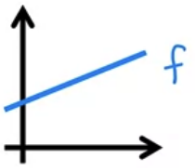


model:

$f_{w,b}(x) = wx + b$

parameters:

w, b



cost function:

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

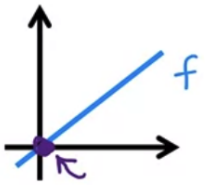
goal:

$\underset{w,b}{\text{minimize}}\, J(w,b)$

simplified

$f_w(x) = wx$ $\quad b = 0$

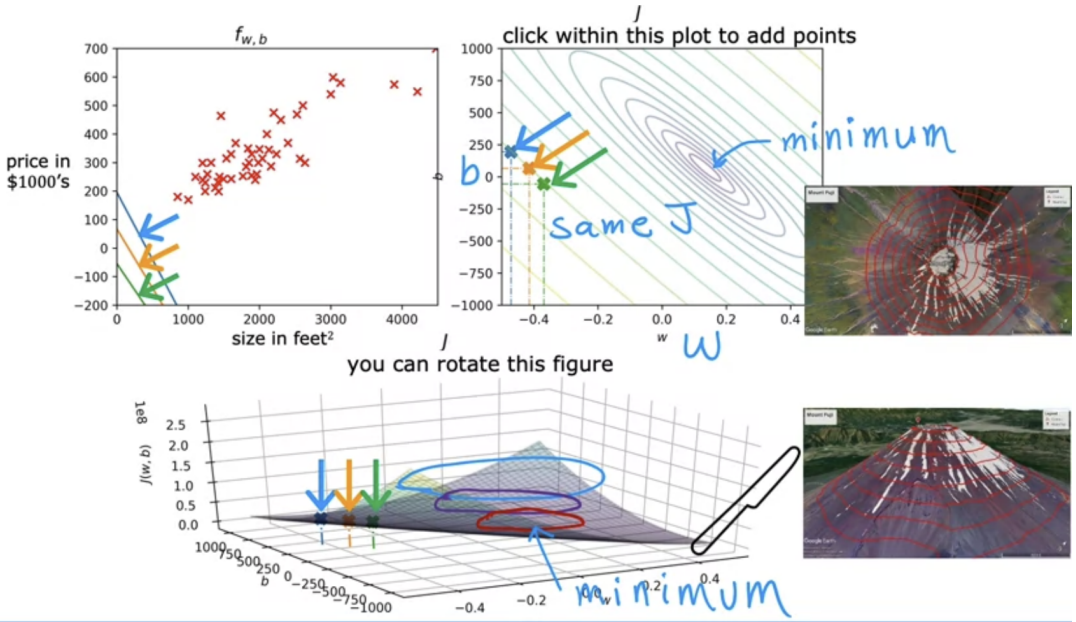
w



$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f_w(x^{(i)}) - y^{(i)})^2$$

$wx^{(i)}$

$\underset{w}{\text{minimize}}\, J(w)$



Have some function $J(w,b)$ for linear regression or any function

Want $\min_{w,b} J(w,b)$ $\min_{w_1,\dots,w_n,b} J(w_1,w_2,\dots,w_n,b)$

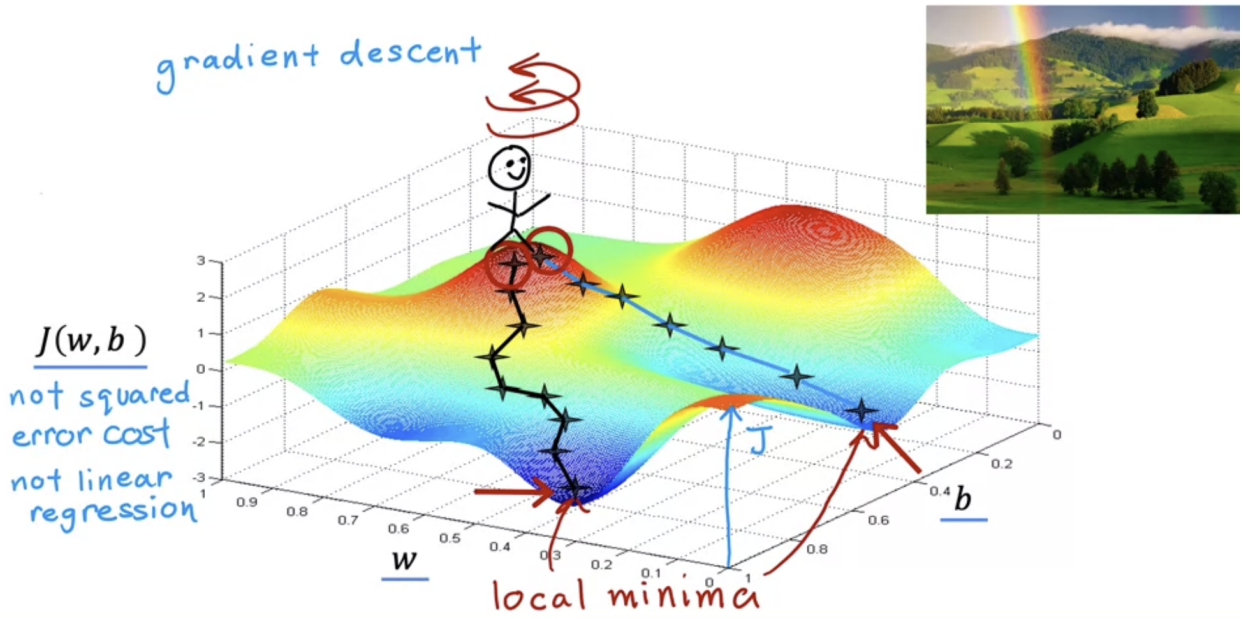
Outline:

- Start with some w,b (set $w=0$, $b=0$)
- Keep changing w,b to reduce $J(w,b)$
- Until we settle at or near a minimum

J not always

may have >1 minimum

Stanford ONLINE DeepLearning.AI Andrew Ng



Stanford ONLINE DeepLearning.AI Andrew Ng

SupervisedMachineLearning-WEEK-1-QUIZ-2

2.3.1. Gradient Descent

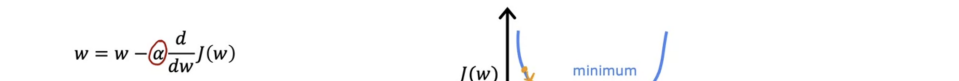
Have some function $J(w,b)$ for linear regression or any function

Want $\min_{w,b} J(w,b)$ $\min_{w_1,\dots,w_n,b} J(w_1,w_2,\dots,w_n,b)$

Outline:

- Start with some w,b (set $w=0$, $b=0$)
- Keep changing w,b to reduce $J(w,b)$
- Until we settle at or near a minimum

J not always



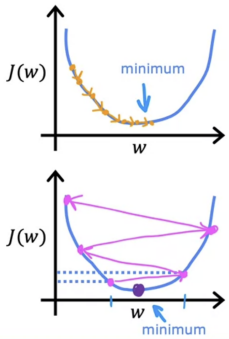
Stanford ONLINE DeepLearning.AI Andrew Ng

$$w = w - \alpha \frac{d}{dw} J(w)$$

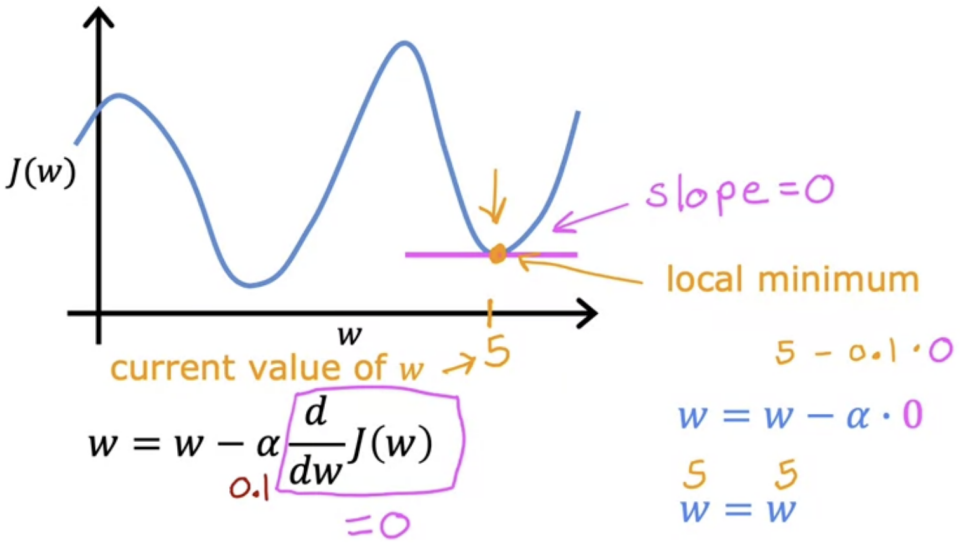
If α is too small...
Gradient descent may be slow.

If α is too large...

Gradient descent may:
- Overshoot, never reach minimum
- Fail to converge, diverge



Stanford ONLINE DeepLearning.AI Andrew Ng



Stanford ONLINE DeepLearning.AI Andrew Ng

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})$$

}

Stanford ONLINE DeepLearning.AI Andrew Ng

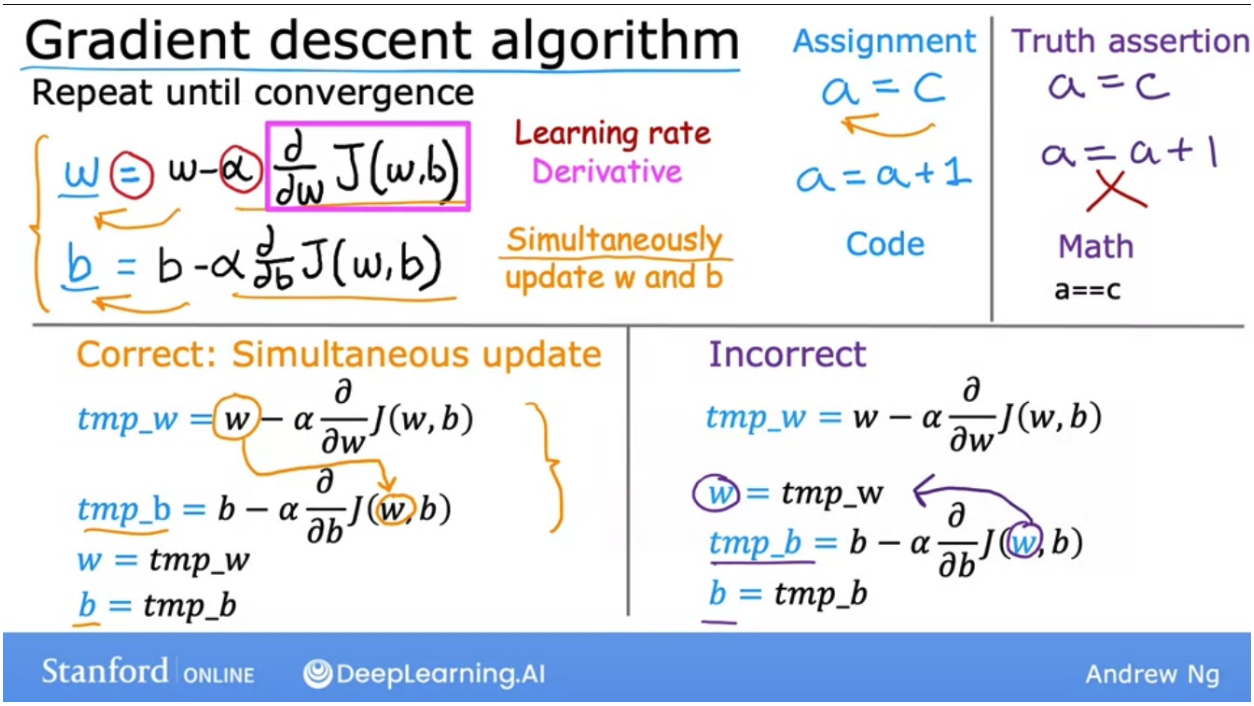
Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) \quad x^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$$

}



1. Gradient descent is an algorithm for finding values of parameters w and b that minimize the cost function J.

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

When $\frac{\partial J(w,b)}{\partial w}$ is a negative number (less than zero), what happens to w after one update step?

- ○ w decreases
- ○ w stays the same
- ◉ w increases.
- ○ It is not possible to tell if w will increase or decrease.

Correct
The learning rate is always a positive number, so if you take W minus a negative number, you end up with a new value for W that is larger (more positive).

2. For linear regression, what is the update step for parameter b?

- ◉ $b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$
- ○ $b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$

Correct
The update step is $b = b - \alpha \frac{\partial J(w,b)}{\partial w}$ where $\frac{\partial J(w,b)}{\partial b}$ can be computed with this expression: $\sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$

Gradient descent summary

So far in this course, you have developed a linear model that predicts $f_{w,b}(x^{(i)})$:

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b \tag{1}$$

In linear regression, you utilize input training data to fit the parameters w,b by minimizing a measure of the error between our predictions $f_{w,b}(x^{(i)})$ and the actual data $y^{(i)}$. The measure is called the $cost$, $J(w,b)$. In training you measure the cost over all of our training samples $x^{(i)}$, $y^{(i)}$

$$J(w,b) = \frac{1}{2m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})^2 \tag{2}$$

In lecture, *gradient descent* was described as:

$$\begin{aligned} &\text{repeat until convergence: } \{ \\ &\quad w = w - \alpha\frac{\partial J(w,b)}{\partial w} \\ &\quad b = b - \alpha\frac{\partial J(w,b)}{\partial b} \\ &\} \end{aligned} \tag{3}$$

where, parameters w, b are updated simultaneously.
The gradient is defined as:

$$\frac{\partial J(w,b)}{\partial w} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)} \tag{4}$$

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)}) \tag{5}$$

Here *simultaniously* means that you calculate the partial derivatives for all the parameters before updating any of the parameters.

Das Gradientenverfahren ist ein allgemeiner Algorithmus zur optimierung, der Optimale Lösungen für eine Vielzahl von Fragestellungen ermitteln kann. Der Grundgedanke beim GV ist, die Parameter iterativ so zu verändern, dass eine Kostenfunktion minimiert wird.

Das Gradientenverfahren berechnet den lokalen Gradienten der Fehlerfunktion in Abhängigkeit vom Parametervektor (omega) und bewegt sich in Richtung eines abfallenden Gradienten. Sobald der Gradient null wird, haben Sie ein Minimum gefunden.

Wenn die Lernrate zu kein ist, muss der Algorithmus viele Iterationen durchlaufen, bevor er konvergiert. Das dauert natürlich sehr lange.

Wenn die Lernrate dagegen zu groß ist, kann es passieren, dass Sie die Täler überspringen und auf der anderen Seite landen, möglicherweise sogar höher als zuvor. Dadurch kann der Algorithmus divergieren, also immer größerer Werte erzeugen und überhaupt keine gute Lösungen finden.

```
#Function to calculate the cost
def compute_cost(x, y, w, b):

    m = x.shape[0]
    cost = 0

    for i in range(m):
        f_wb = w * x[i] + b
        cost = cost + (f_wb - y[i])**2
    total_cost = 1 / (2 * m) * cost

    return total_cost
```

```
def compute_gradient(x, y, w, b):
    """
    Computes the gradient for linear regression
    Args:
      x (ndarray (m,)): Data, m examples
      y (ndarray (m,)): target values
      w,b (scalar)    : model parameters
    Returns
      dj_dw (scalar): The gradient of the cost w.r.t. the parame
      dj_db (scalar): The gradient of the cost w.r.t. the parame
     """

    # Number of training examples
    m = x.shape[0]
```

```
    dj_dw = 0
    dj_db = 0

    for i in range(m):
        f_wb = w * x[i] + b
        dj_dw_i = (f_wb - y[i]) * x[i]
        dj_db_i = f_wb - y[i]
        dj_db += dj_db_i
        dj_dw += dj_dw_i
    dj_dw = dj_dw / m
    dj_db = dj_db / m

    return dj_dw, dj_db
```

```
def gradient_descent(x, y, w_in, b_in, alpha, num_iters, cost_fu
    """
    Performs gradient descent to fit w,b. Updates w,b by taking
    num_iters gradient steps with learning rate alpha

    Args:
      x (ndarray (m,))  : Data, m examples
      y (ndarray (m,))  : target values
      w_in,b_in (scalar): initial values of model parameters
      alpha (float):     Learning rate
      num_iters (int):   number of iterations to run gradient de
      cost_function:     function to call to produce cost
      gradient_function: function to call to produce gradient

    Returns:
      w (scalar): Updated value of parameter after running gradi
      b (scalar): Updated value of parameter after running gradi
      J_history (List): History of cost values
      p_history (list): History of parameters [w,b]
      """
```

```
    # An array to store cost J and w's at each iteration primar
    J_history = []
    p_history = []
    b = b_in
    w = w_in

    for i in range(num_iters):
        # Calculate the gradient and update the parameters using
        dj_dw, dj_db = gradient_function(x, y, w , b)

        # Update Parameters using equation (3) above
        b = b - alpha * dj_db
        w = w - alpha * dj_dw

        # Save cost J at each iteration
        if i<100000:      # prevent resource exhaustion
            J_history.append( cost_function(x, y, w , b))
            p_history.append([w,b])
        # Print cost every at intervals 10 times or as many iter
        if i% math.ceil(num_iters/10) == 0:
            print(f"Iteration {i:4}: Cost {J_history[-1]:0.2e} "
                  f"dj_dw: {dj_dw: 0.3e}, dj_db: {dj_db: 0.3e}
                  f"w: {w: 0.3e}, b:{b: 0.5e}")

    return w, b, J_history, p_history #return w and J,w history
```

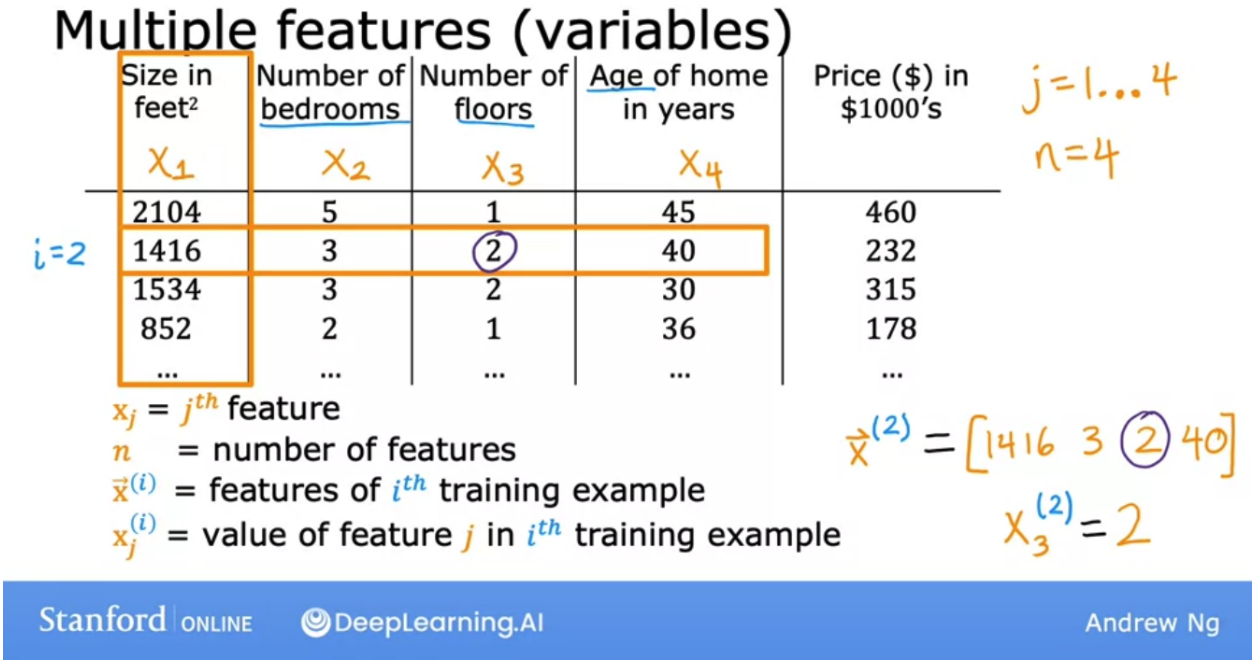
MSE ist als Kostenfunktion eines linearen Regressionsmodells eine konvexe Funktion. Das bedeutet, wenn Sie zwei beliebige Punkte auf der Kurve auswählen, liegt die lineare Verbindung zwischen diesen beiden niemals unter der Kurve. Das impliziert, dass es keine lokalen Minima gibt, nur ein globales Minimum. Sie ist auch eine stetige Funktion mit einer Steigung, die sich niemals abrupt ändert. Diese zwei Umsände haben eine wichtige Konsequenz: Mit dem Gradientenverfahren kann man sich dem globalen Minimum beliebig annähern (wenn Sie lange genug warten und die Lernrate nicht zu groß ist).

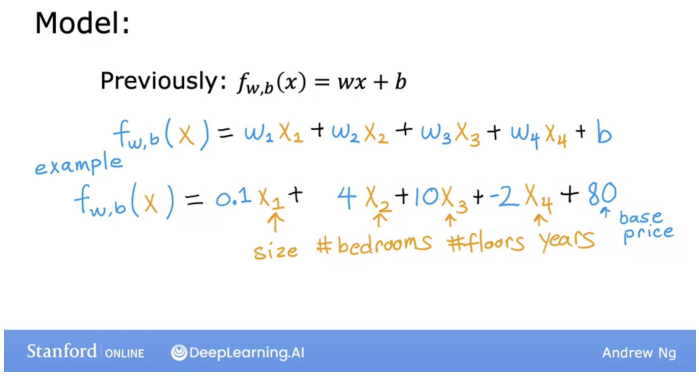
Die Kostenfunktion hat also die Form einer Schüssel. Sind die Merkmale sehr unterschiedlich skaliert, kann es aber eine längliche Schüssel sein.

Sie fragen sich vielleicht, wie man die Anzahl der Epochen bestimmen soll. Wenn diese zu gering ist, werden Sie beim Anhalten des Algorithmus noch immer weit von der optimalen Lösung entfernt sein. Ist sie aber zu hoch, verschwenden Sie Zeit, während sich die Modellparameter nicht mehr verändern. Eine einfache Lösung ist, die Anzahl Iterationen auf einen sehr großen Wert zu setzen, aber den Algorithmus anzuhalten, sobald der Gradientenvektor winzig klein wird — denn das passiert, wenn das Gradientenverfahren das Minimum (beinahe) erreicht hat. Der Gradientenvektor wird als winzig betrachtet, wenn sein Betrag kleiner als eine sehr kleine Zahl x wird, was man auch als Toleranz bezeichnet.

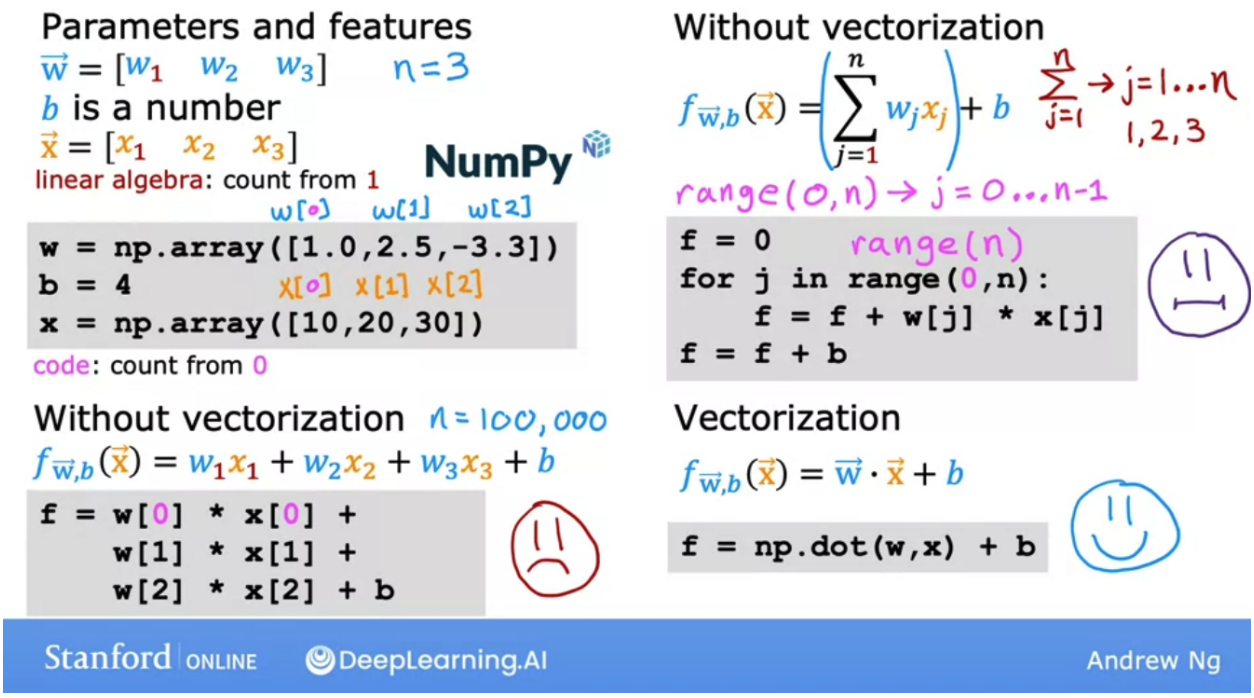
SupervisedMachineLearning-WEEK-1-QUIZ-3

2.4. Multiple Linear Regression





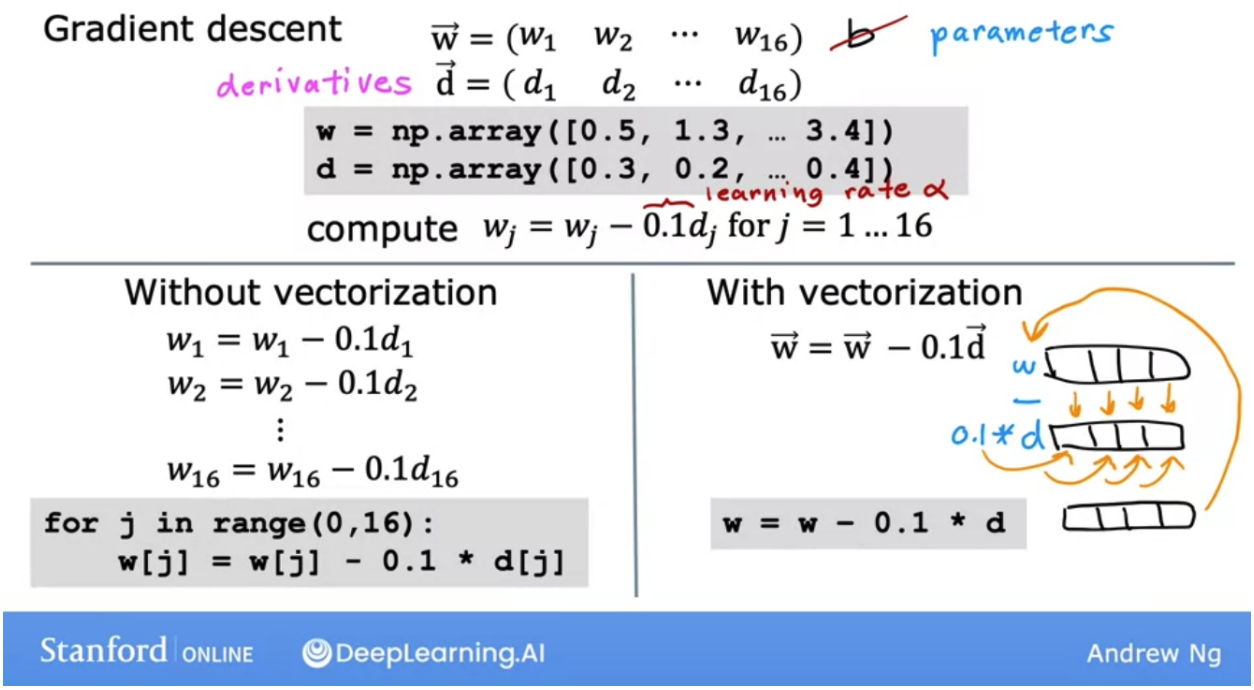
2.4.1. Vectorization



Codes with vectorization perform fast than codes without vectorization.

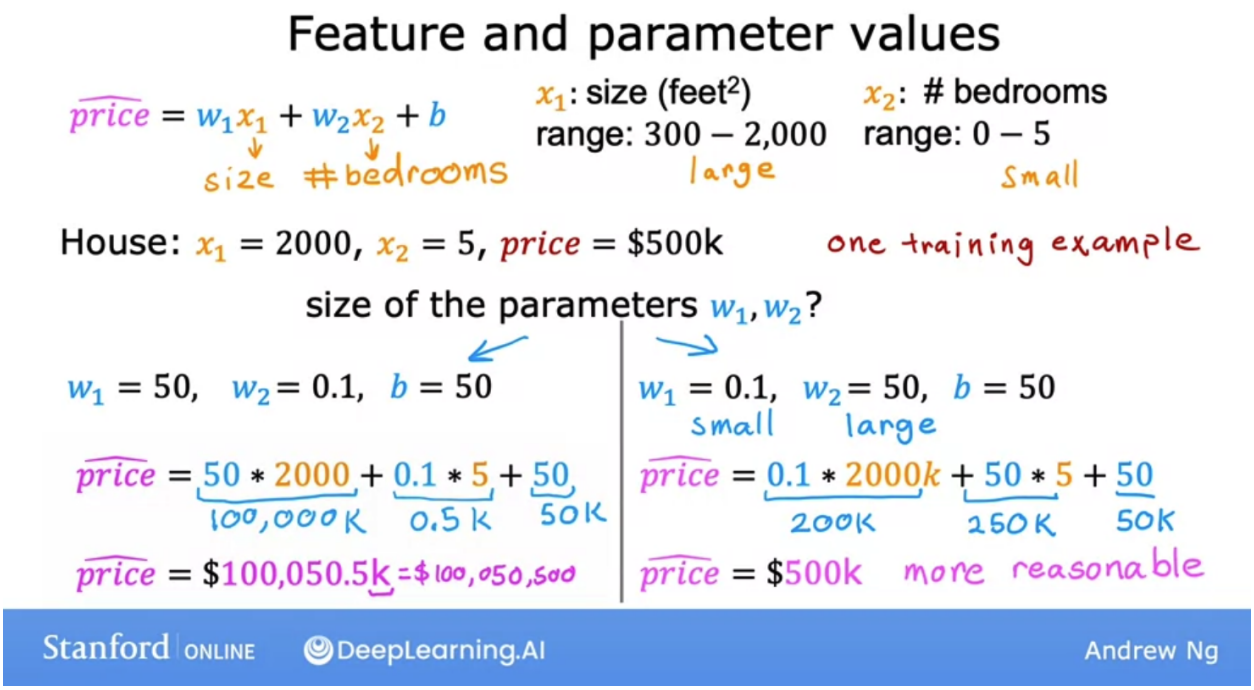
2.4.1.1. Vector Dot Product

The dot product multiplies the values in two vectors element-wise and then sums the result. Vector dot product requires the dimensions of the two vectors to be the same.

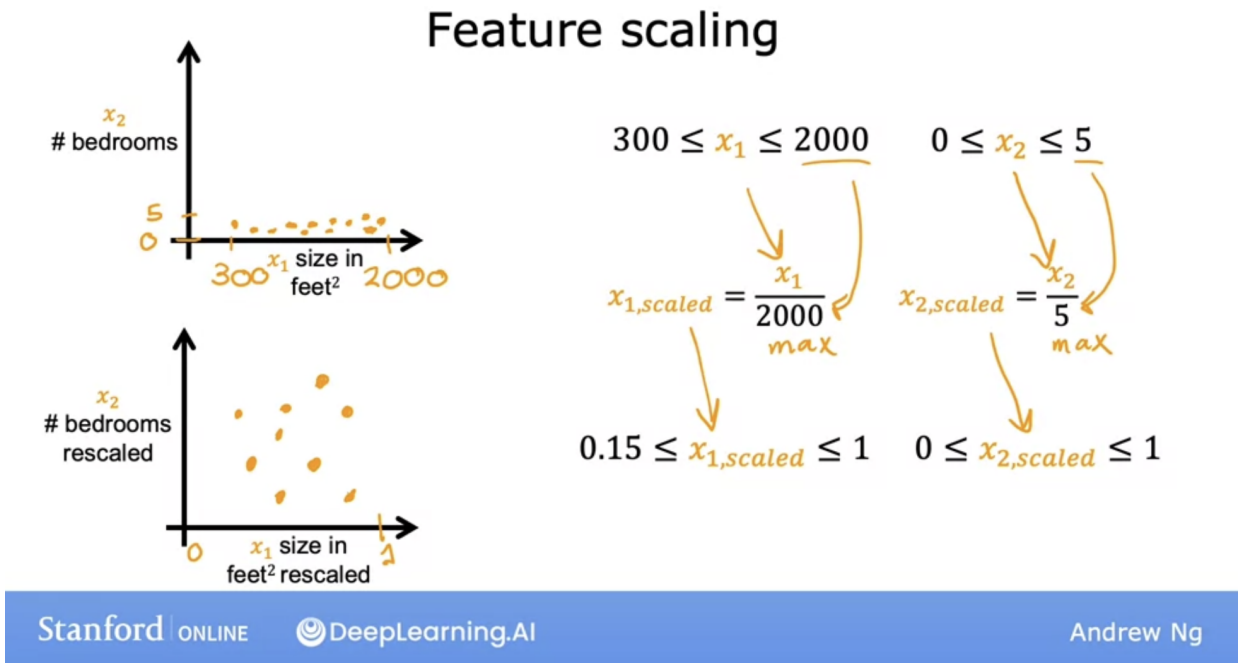


SupervisedMachineLearning-WEEK-2-QUIZ-1

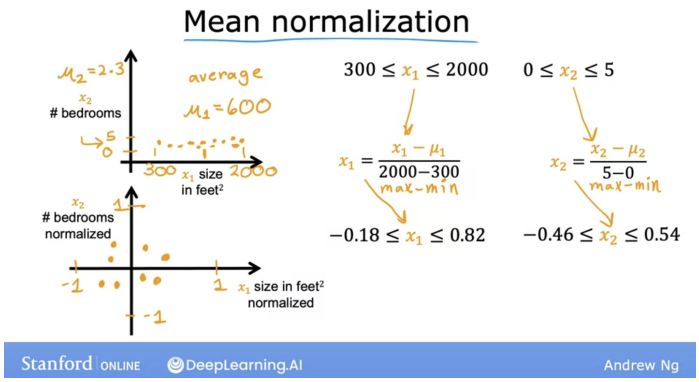
2.5. Feature Scaling



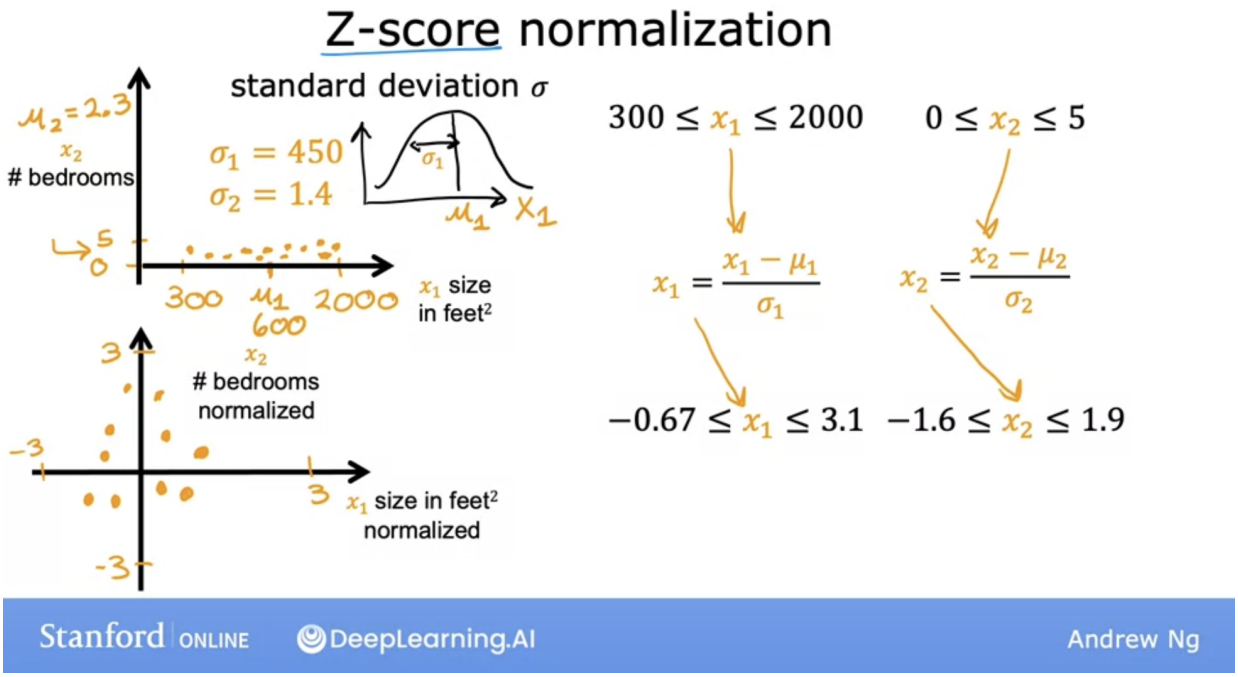
— Divide each number by the maximum number.



— Mean normalization (Min-max scaling): find the average (mean) and then minus each number by the calculated mean and then divide them by (max - min).



— Z-score normalization: calculate the standard deviation



Feature scaling

aim for about $-1 \le x_j \le 1$ for each feature x_j

$-3 \le x_j \le 3$

$-0.3 \le x_j \le 0.3$

} acceptable ranges

$0 \le x_1 \le 3$ okay, no rescaling

$-2 \le x_2 \le 0.5$ okay, no rescaling

$-100 \le x_3 \le 100$ too large → rescale

$-0.001 \le x_4 \le 0.001$ too small → rescale

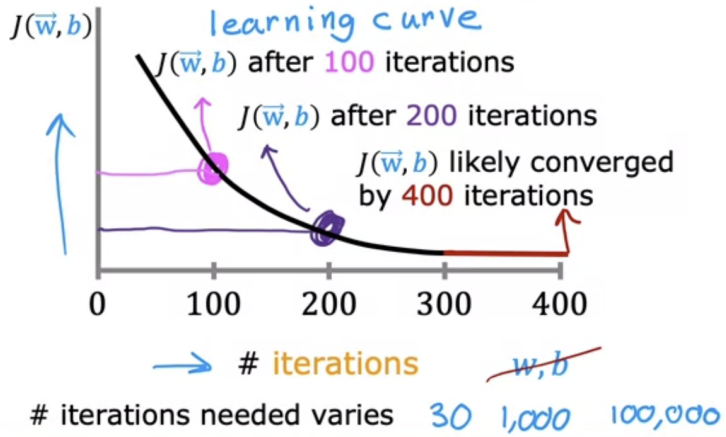
$98.6 \le x_5 \le 105$ too large →

Stanford ONLINE DeepLearning.AI Andrew Ng

Make sure gradient descent is working correctly

objective: $\min_{\vec{w},b} J(\vec{w},b)$

$J(\vec{w},b)$ should decrease after every iteration



iterations needed varies 30 1,000 100,000

Automatic convergence test

Let ε "epsilon" be 10^{-3}. 0.001

If $J(\vec{w},b)$ decreases by $\le \varepsilon$ in one iteration, declare convergence.

(found parameters $\vec{w},b$ to get close to global minimum)

Stanford ONLINE DeepLearning.AI Andrew Ng

2.6. Identify problem with gradient descent

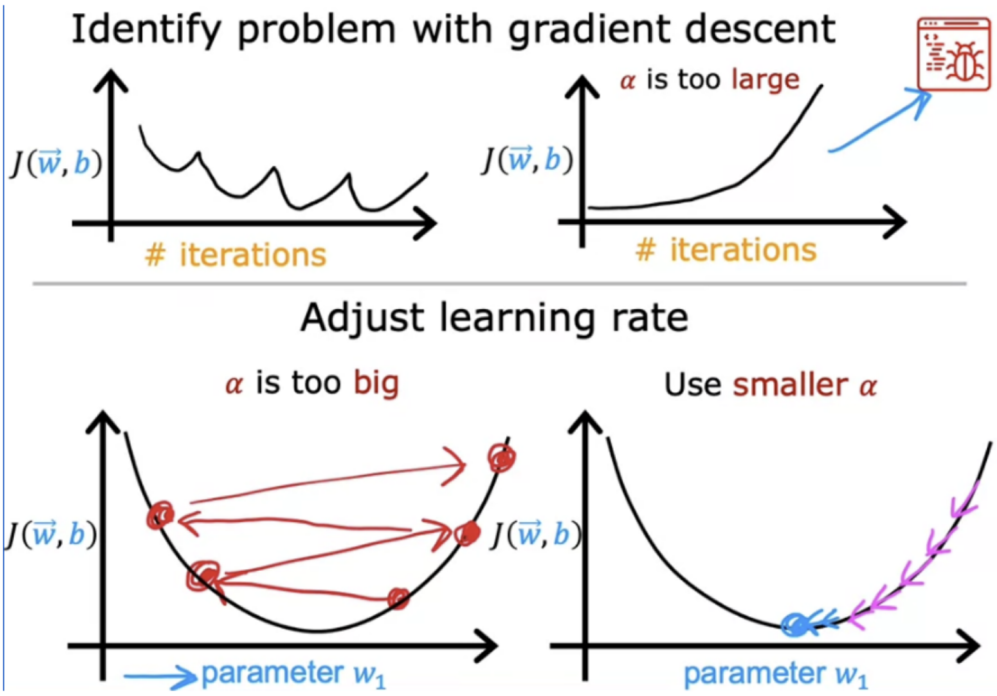
▼ Gradient Descent should consistently decrease the loss function, provided everything is set up correctly. If loss is bouncing up and down, it could be due to one or more of the following reasons:

- ▼ Learning Rate is too large

  A large learning rate can cause the gradient updates to overshoot the optimal point, making the loss oscillate or even diverge.

- ▼ Bug in the code

  Errors in implementing the gradient computation, backpropagation, or updating the weights.



Gradient Descent With Multiple Variables

Here are the equations you developed in the last lab on gradient descent for multiple variables.:

$$\begin{aligned}&\text{repeat until convergence: } \{\\ &\quad w_j := w_j - \alpha \frac{\partial J(\mathbf{w},b)}{\partial w_j} \qquad \text{for j = 0..n-1} \qquad (1)\\ &\quad b := b - \alpha \frac{\partial J(\mathbf{w},b)}{\partial b}\\ &\}\end{aligned}$$

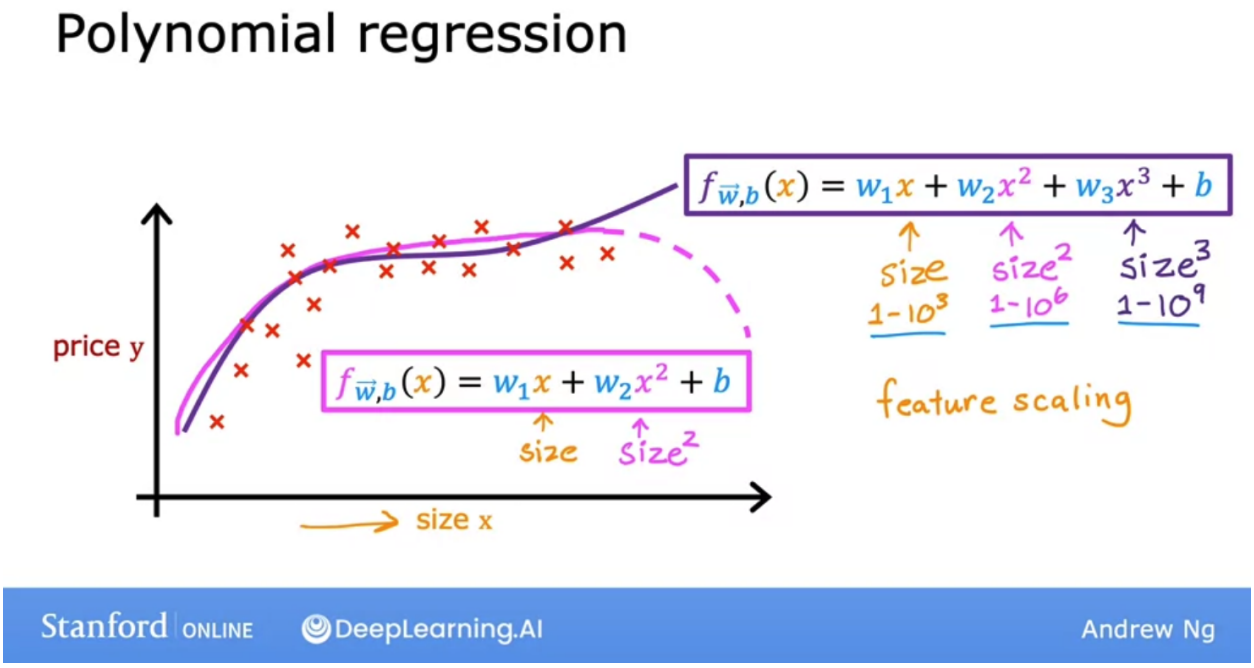
where, n is the number of features, parameters w_j, b, are updated simultaneously and where

$$\frac{\partial J(\mathbf{w},b)}{\partial w_j} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})x_j^{(i)} \qquad (2)$$

$$\frac{\partial J(\mathbf{w},b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) \qquad (3)$$

- m is the number of training examples in the data set
- $f_{\mathbf{w},b}(\mathbf{x}^{(i)})$ is the model's prediction, while $y^{(i)}$ is the target value

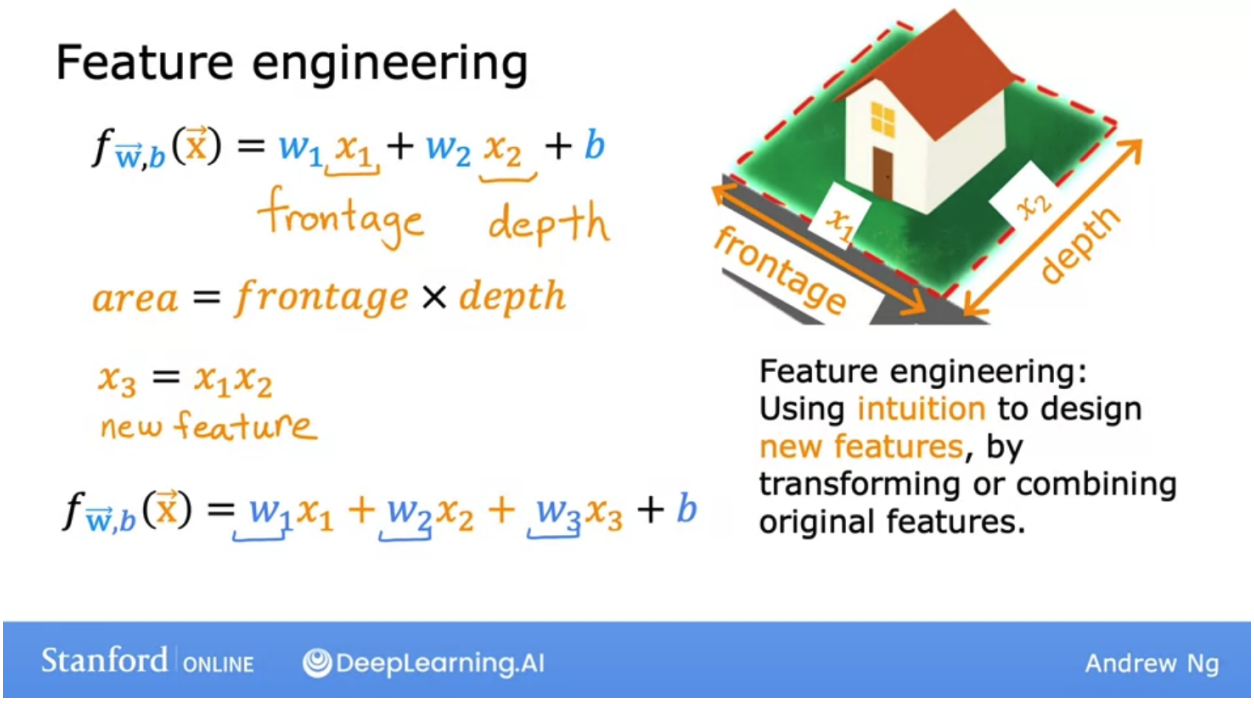
2.7. Polynomial Regression



SupervisedMachineLearning-WEEK-2-QUIZ-2

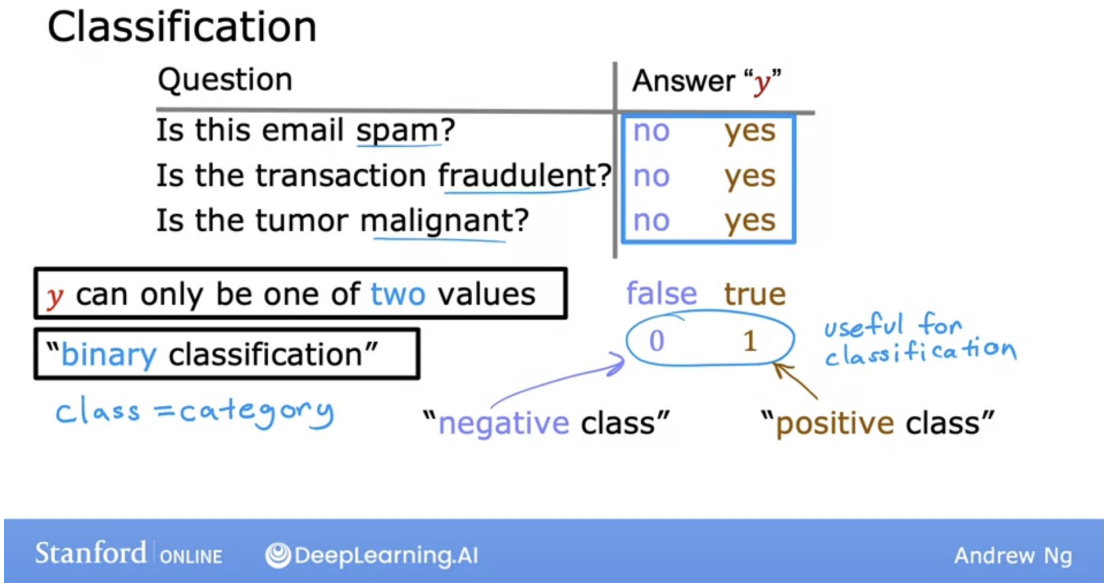
III. Feature engineering

Feature engineering: Using intuition to design new features, by transforming or combining original features.

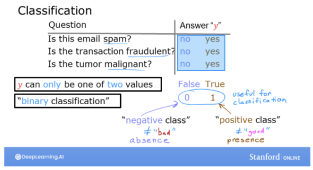


WEEK2_ASSIGNMENT_VADHA_SAMEDY_HUN.ipynb

4. Classification (Logistic Regression)



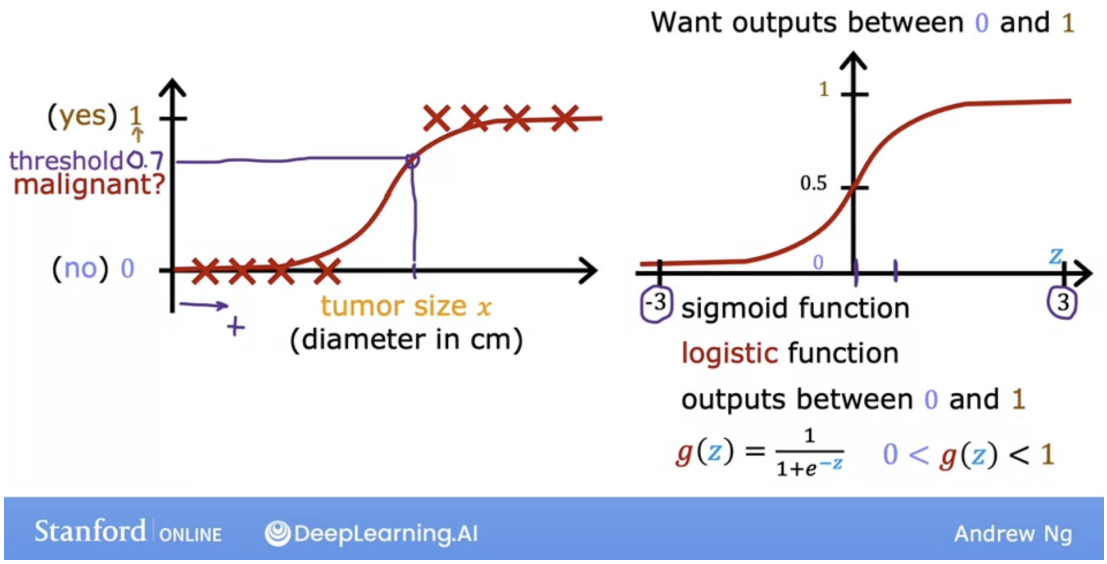
Classification Problems



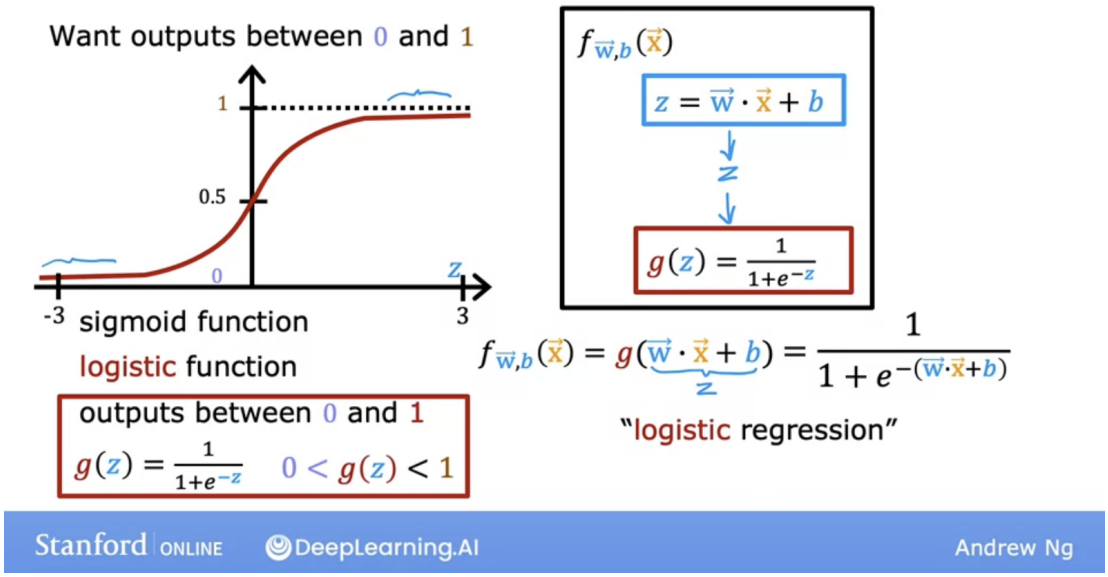
Examples of classification problems are things like: identifying email as Spam or Not Spam or determining if a tumor is malignant or benign. In particular, these are examples of *binary* classification where there are two possible outcomes. Outcomes can be described in pairs of 'positive'/'negative' such as 'yes'/'no, 'true'/'false' or '1'/'0'.

Plots of classification data sets often use symbols to indicate the outcome of an example. In the plots below, 'X' is used to represent the positive values while 'O' represents negative outcomes.

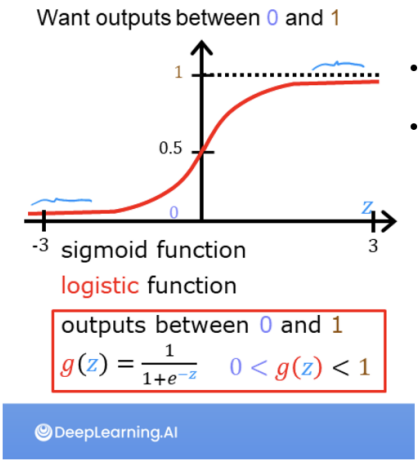
4.1. Logistic regression



Sigmoid function (logistic function) → output 0 and 1.



Sigmoid or Logistic Function



As discussed in the lecture videos, for a classification task, we can start by using our linear regression model, $f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \mathbf{w} \cdot \mathbf{x}^{(i)} + b$, to predict y given x.

- However, we would like the predictions of our classification model to be between 0 and 1 since our output variable y is either 0 or 1.
- This can be accomplished by using a "sigmoid function" which maps all input values to values between 0 and 1.

Let's implement the sigmoid function and see this for ourselves.

Formula for Sigmoid function

The formula for a sigmoid function is as follows -

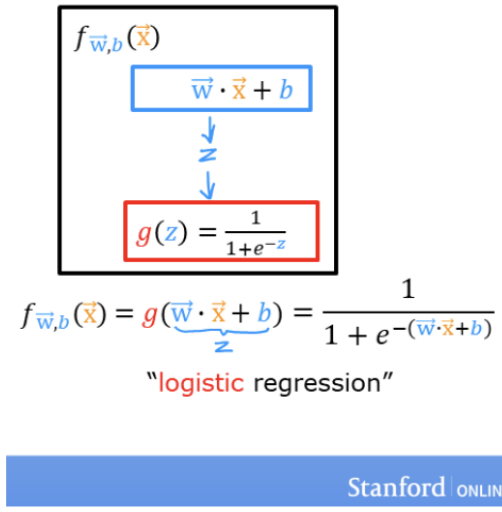
$$g(z) = \frac{1}{1+e^{-z}} \tag{1}$$

In the case of logistic regression, z (the input to the sigmoid function), is the output of a linear regression model.

- In the case of a single example, z is scalar.
- in the case of multiple examples, z may be a vector consisting of m values, one for each example.
- The implementation of the sigmoid function should cover both of these potential input formats. Let's implement this in Python.

Logistic Regression: a logistic regression model applies the sigmoid to the familiar linear regression model as shown:

Logistic Regression

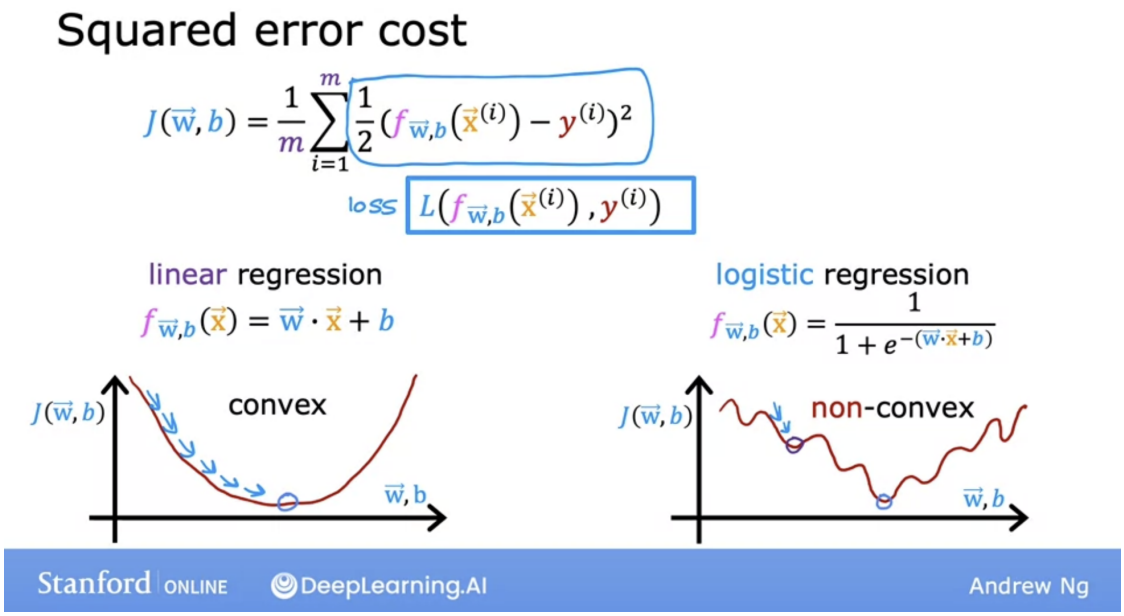


A logistic regression model applies the sigmoid to the familiar linear regression model as shown below:

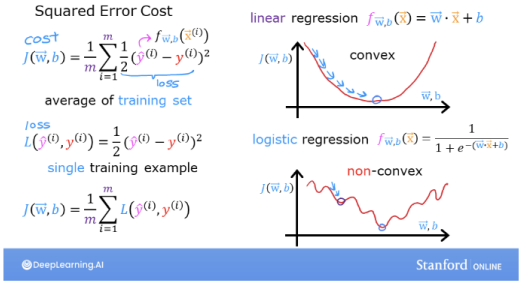
$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = g(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \tag{2}$$

where

$$g(z) = \frac{1}{1+e^{-z}} \tag{3}$$



Squared error for logistic regression?



Recall for **Linear** Regression we have used the **squared error cost function**: The equation for the squared error cost with one variable is:

$$J(w, b) = \frac{1}{2m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})^2 \quad (1)$$

where

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b \quad (2)$$

For the simplified loss function:

$$L(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)}) = -y^{(i)}log(f_{\mathbf{w},b}(\mathbf{x}^{(i)})) - (1 - y^{(i)})log(1 - f_{\mathbf{w},b}(\mathbf{x}^{(i)}))$$

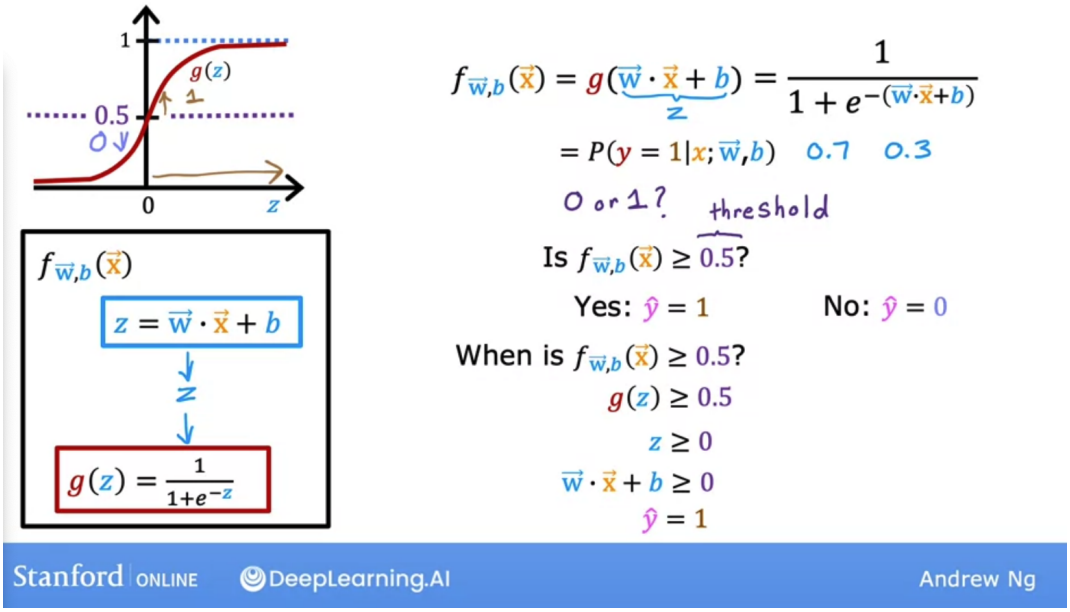
if the target $y^{(i)} = 1$, then what does this expression simplify to?

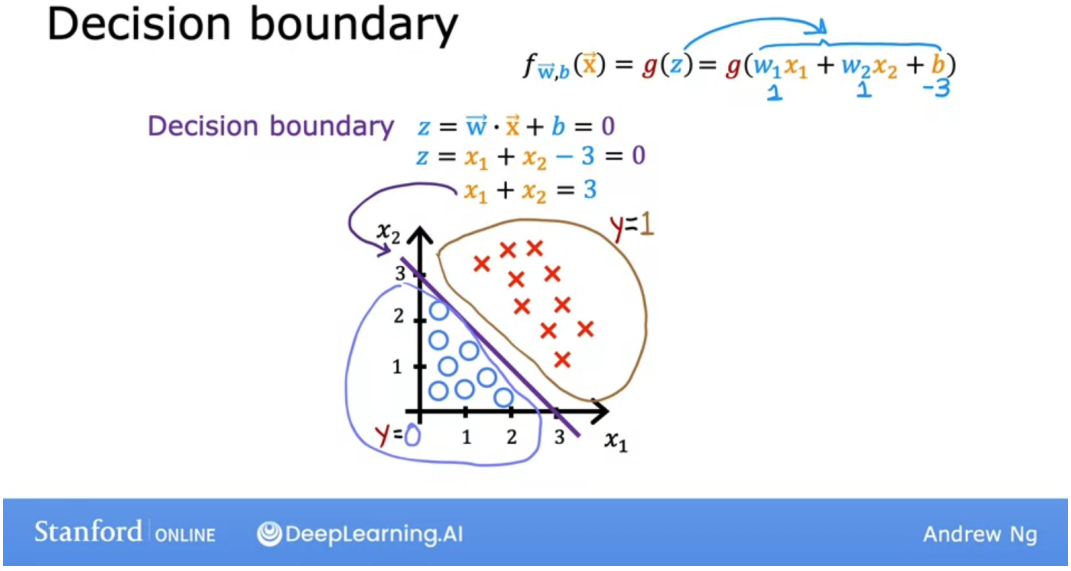
- $-log(1 - f_{\vec{\mathbf{w}},b}(\mathbf{x}^{(i)}))$
- $-log(f_{\vec{w},b}(\mathbf{x}^{(i)}))$

Correct
The second term of the expression is reduced to zero when the target equals 1.

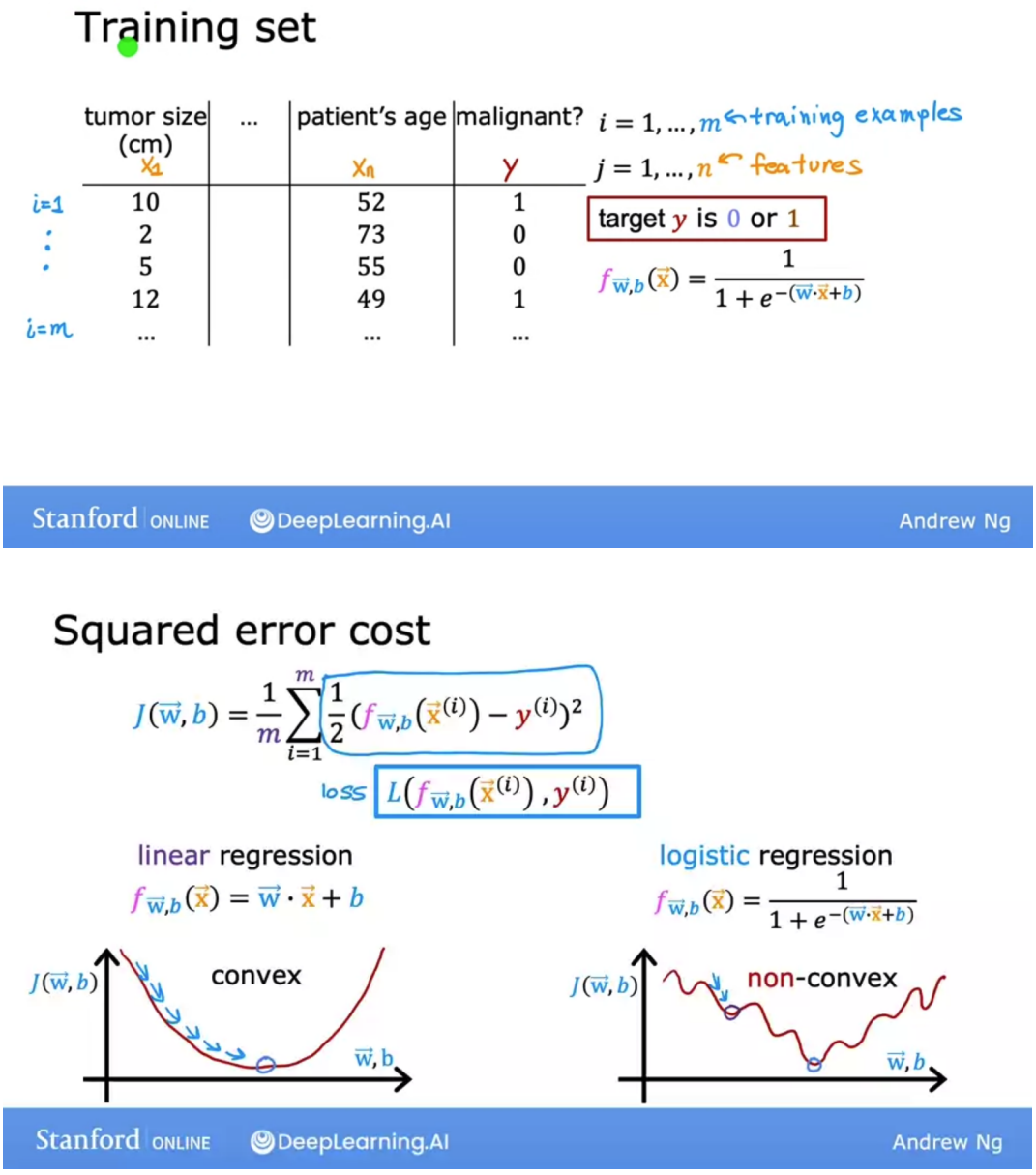
SupervisedMachineLearning-WEEK-3-QUIZ-1

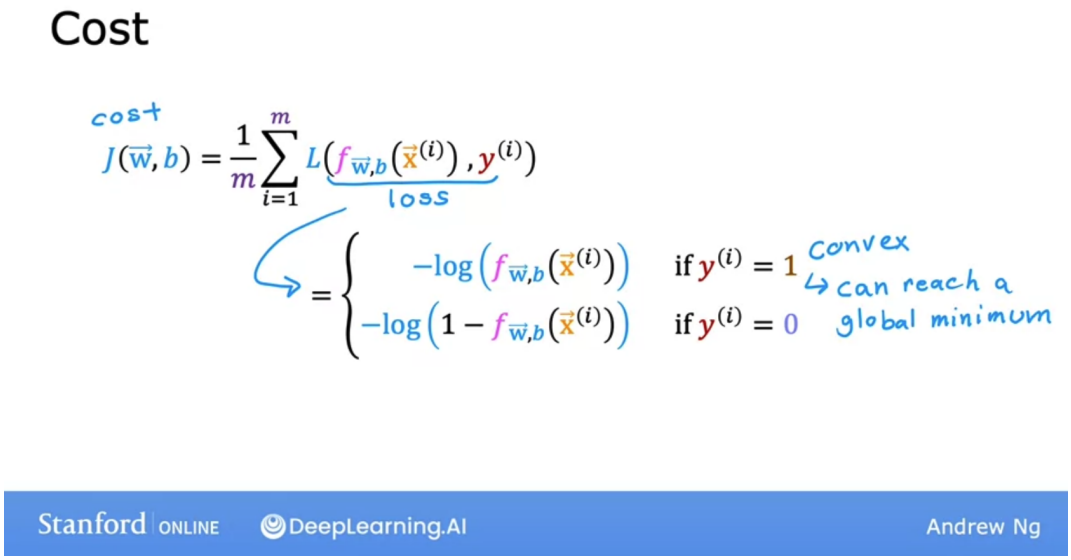
4.1.1. Decision boundary





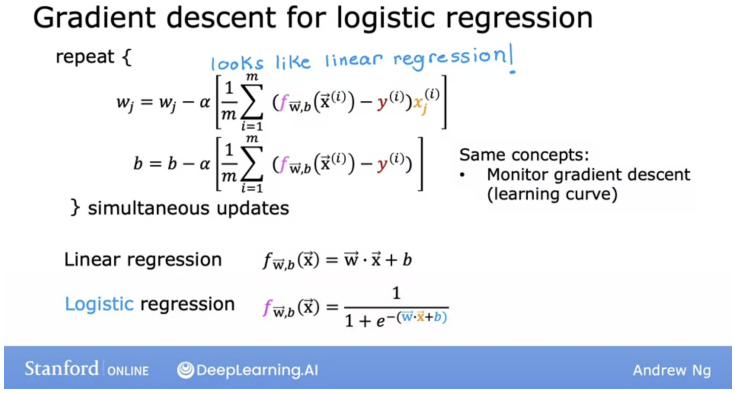
4.1.2. Cost function

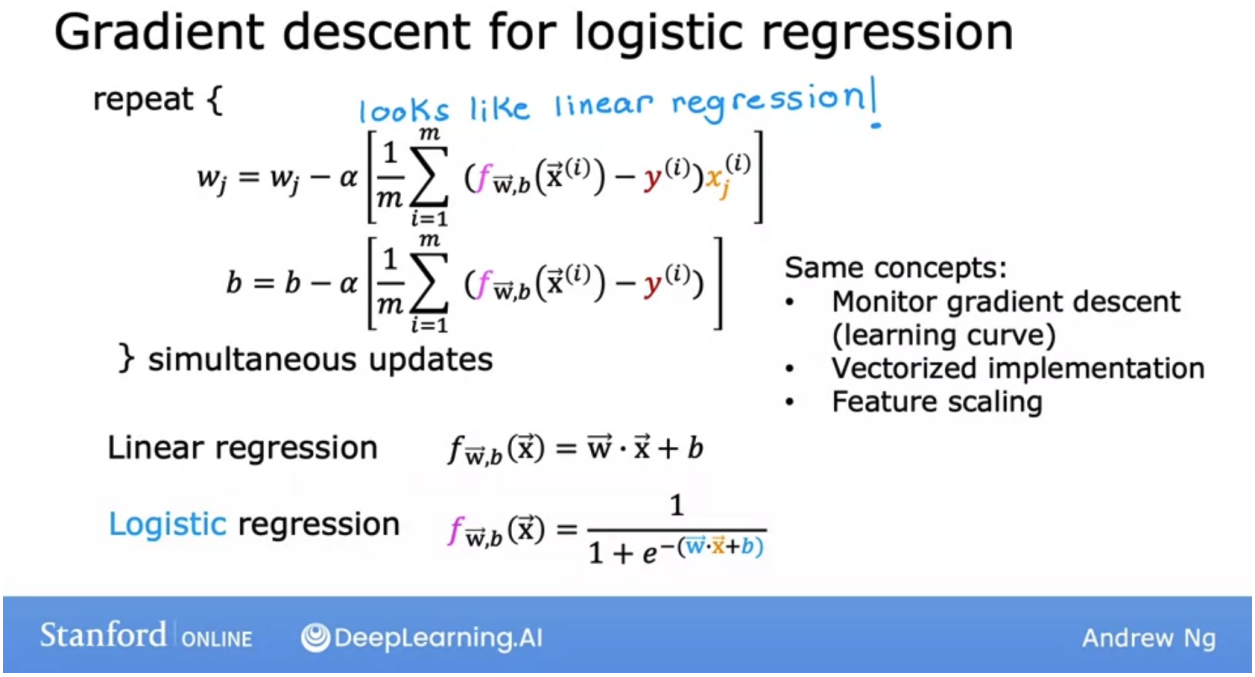




SupervisedMachineLearning-WEEK-3-QUIZ-2

4.2. Gradient descent for logistic regression





Logistic Gradient Descent

Recall the gradient descent algorithm utilizes the gradient calculation:

repeat until convergence: {

$$w_j = w_j - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial w_j} \qquad \text{for j := 0..n-1}$$

$$b = b - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial b}$$

}

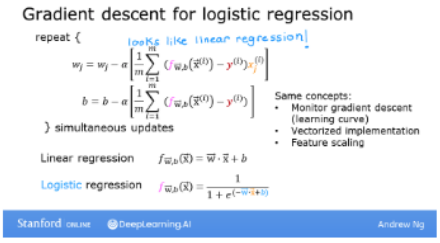
Where each iteration performs simultaneous updates on w_j for all j, where

$$\frac{\partial J(\mathbf{w}, b)}{\partial w_j} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})x_j^{(i)} \quad (2)$$

$$\frac{\partial J(\mathbf{w}, b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) \quad (3)$$

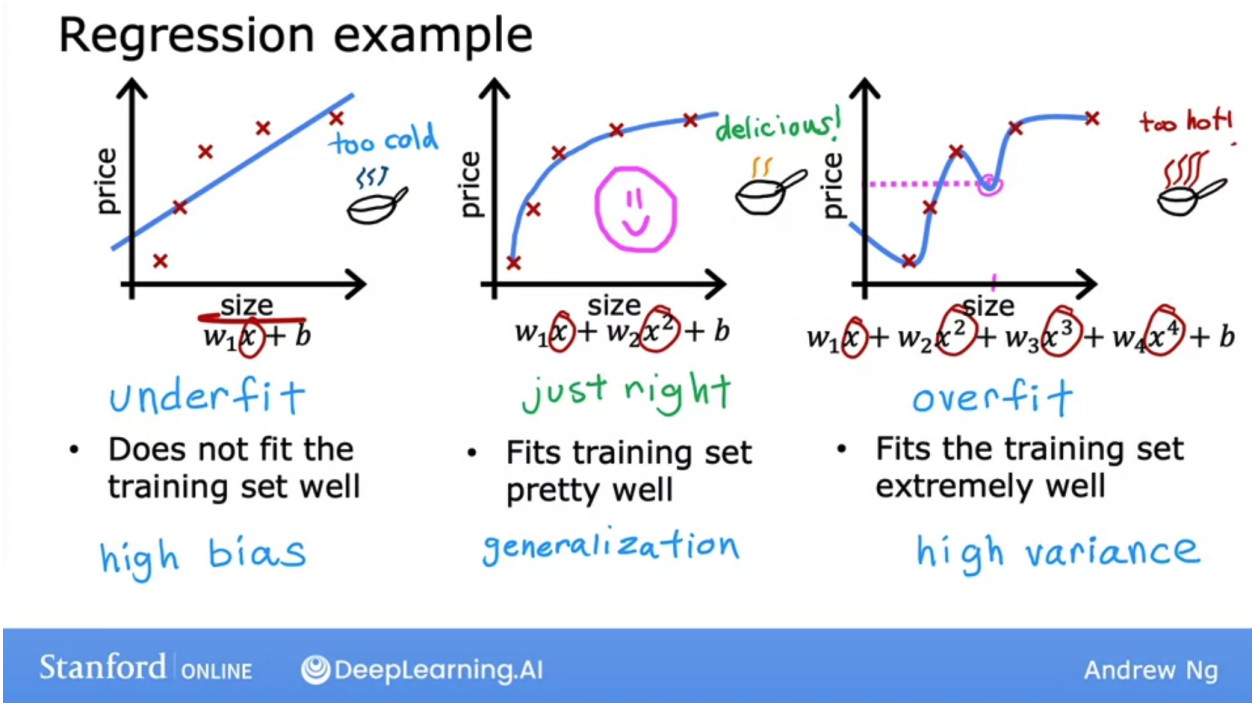
- m is the number of training examples in the data set
- $f_{\mathbf{w},b}(x^{(i)})$ is the model's prediction, while $y^{(i)}$ is the target
- For a logistic regression model
  $z = \mathbf{w} \cdot \mathbf{x} + b$
  $f_{\mathbf{w},b}(x) = g(z)$
  where $g(z)$ is the sigmoid function:
  $g(z) = \frac{1}{1+e^{-z}}$

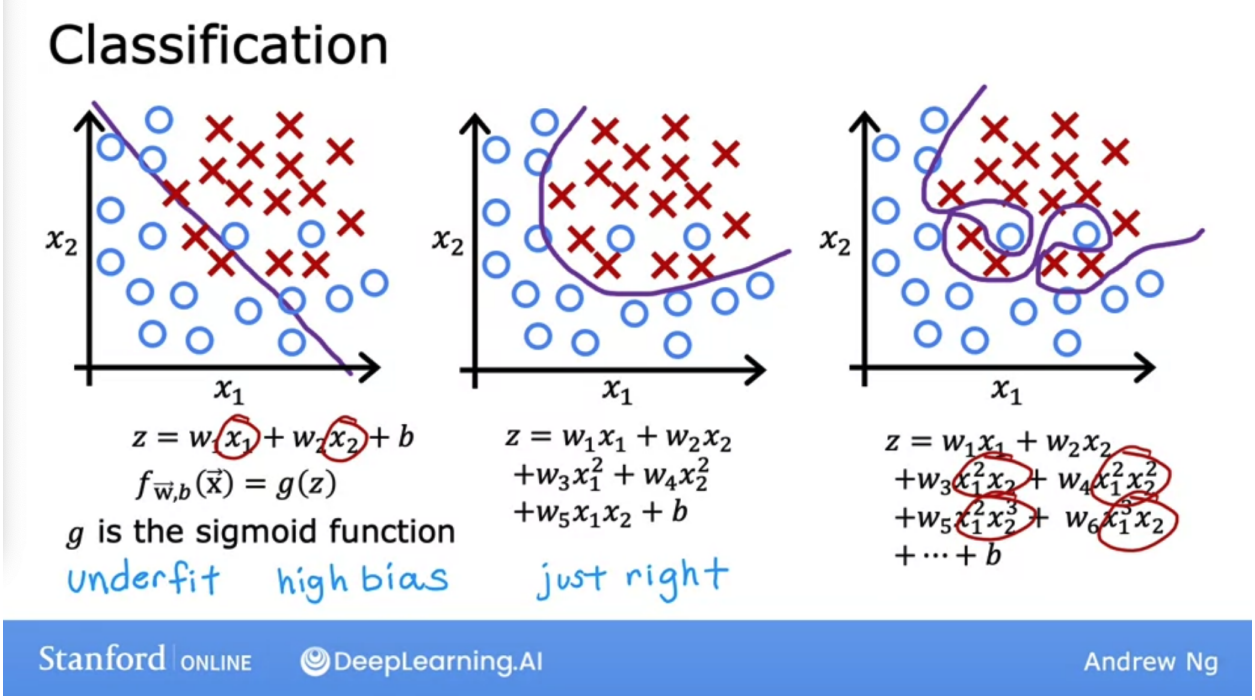
(1)



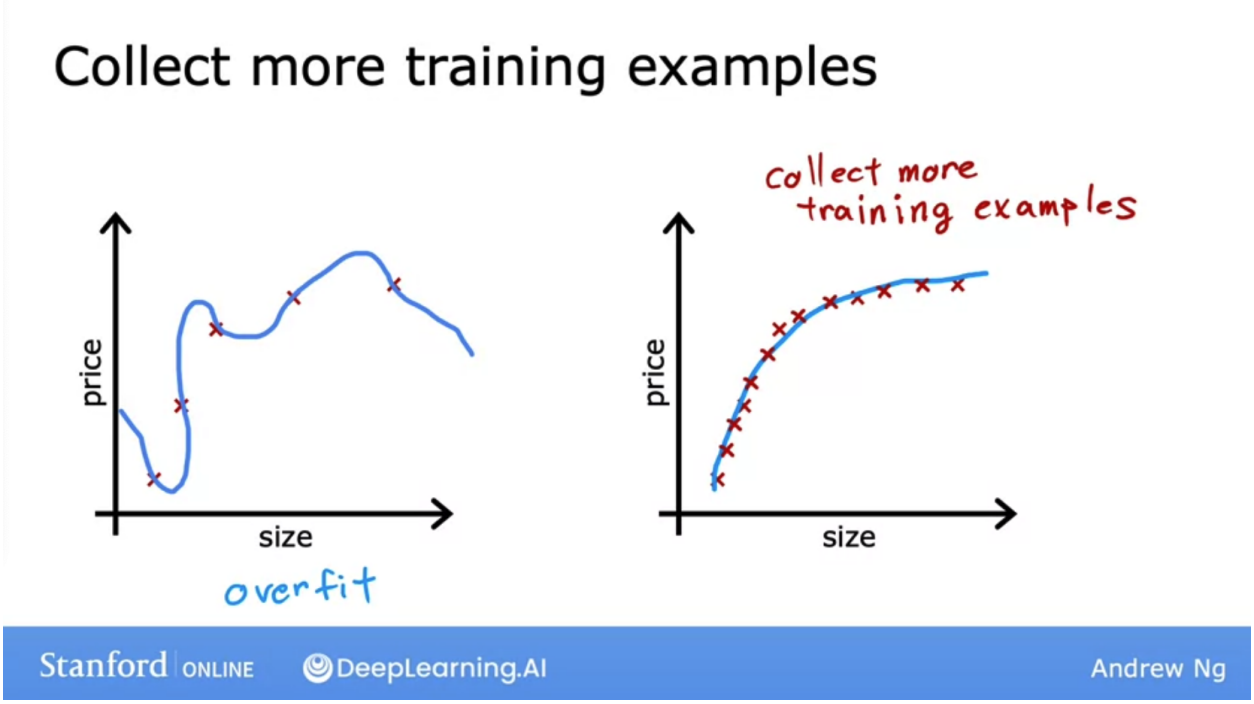
SupervisedMachineLearning-WEEK-3-QUIZ-3

4.3. Overfitting

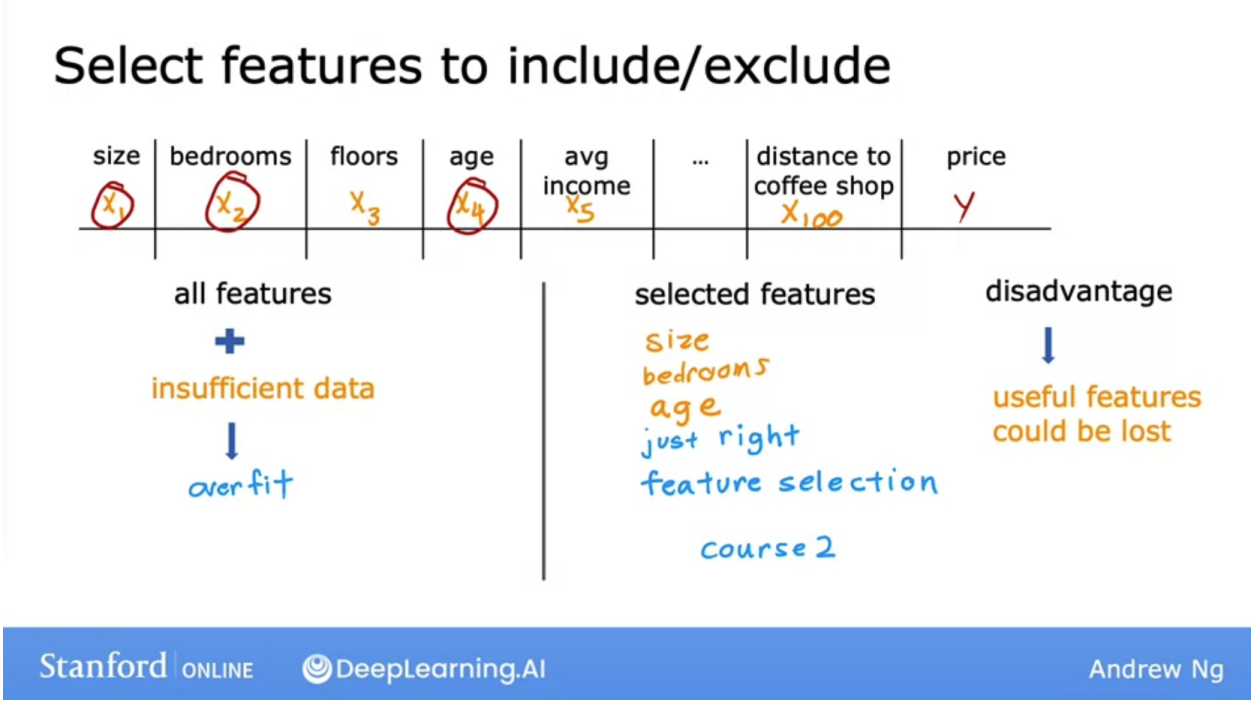




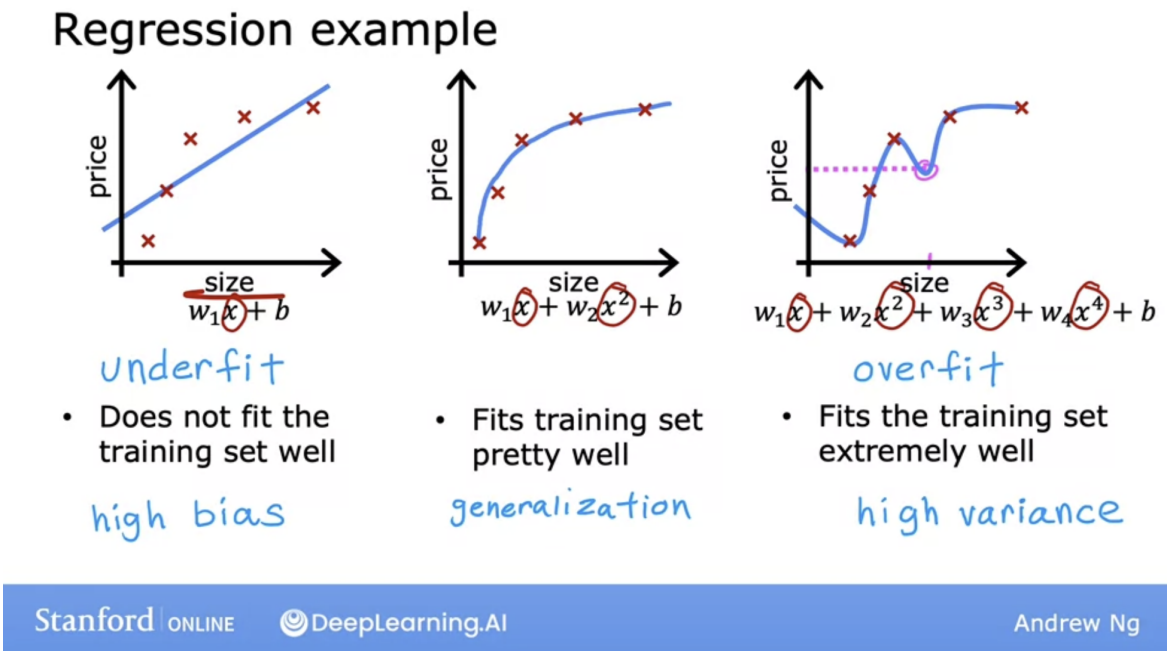
4.3.1. More data to reduce Overfitting

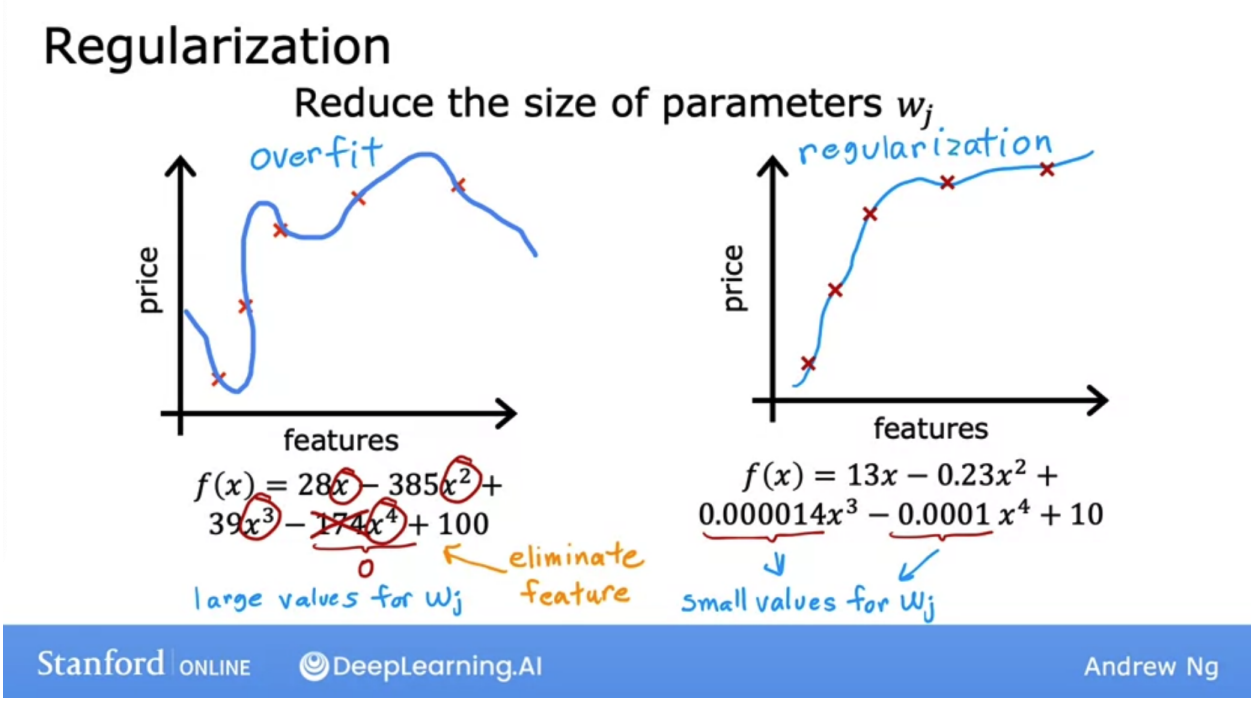


4.3.2. Select feature(s) to include/exclude



4.3.3. Regularization to Reduce Overfitting





4.3.4. Solutions

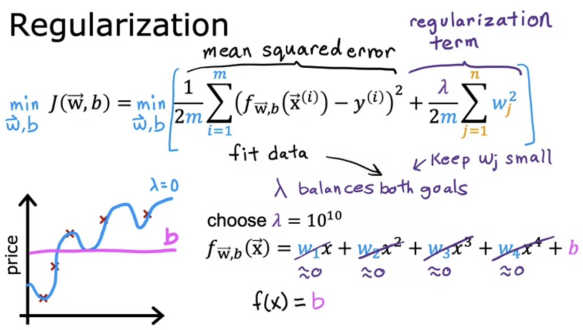
To reduce overfitting:

— Gather more data.

— Select features to include/exclude (just right, feature selection)

— Regularization (penalized): reduce the size of parameters wj (either eliminate a feature (that has big impact) or give small values)

4.3.5. Regularization



How to choose lambda?

If lambda too large → underfit.

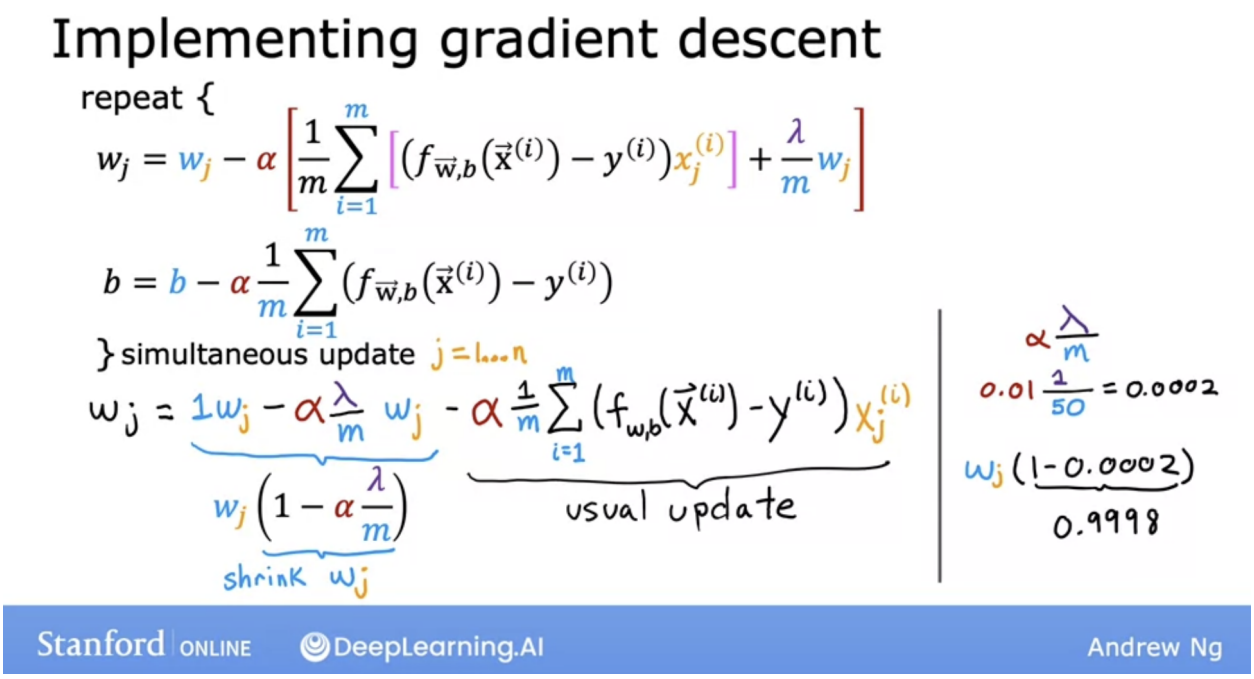
If lambda too small → overfit.

Lambda value needs to appropriately balance the trade-off of minimizing the MSE and keeping the parameter small.

For a model that includes the regularization parameter λ (lambda), increasing λ will tend to…

- ○ Increase the size of parameter b.
- ◉ Decrease the size of parameters $w_1, w_2, \ldots w_n$.
- ○ Increases the size of the parameters $w_1, w_2, \ldots w_n$
- ○ Decrease the size of the parameter b.

Correct
Increasing the regularization parameter $lambda$ reduces overfitting by reducing the size of the parameters. For some parameters that are near zero, this reduces the effect of the associated features.



For regularized **logistic** regression, how do the gradient descent update steps compare to the steps for linear regression?

- (●) They look very similar, but the $f(x)$ is not the same.
- ( ) They are identical

Correct
For logistic regression, $f(x)$ is the sigmoid (logistic) function, whereas for linear regression, $f(x)$ is a linear function.

Regularization

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b} \left[\frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2 \right]$$

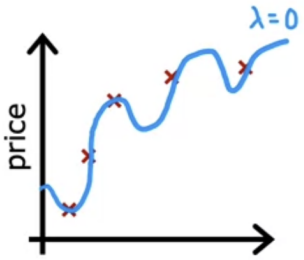
mean squared error

regularization term

fit data

Keep w_j small

λ balances both goals



choose $\lambda = 10^{10}$

$$f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$$

Implementing gradient descent

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left[\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)$$

} simultaneous update $j = 1 \ldots n$

$$w_j = 1 w_j - \alpha \frac{\lambda}{m} w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

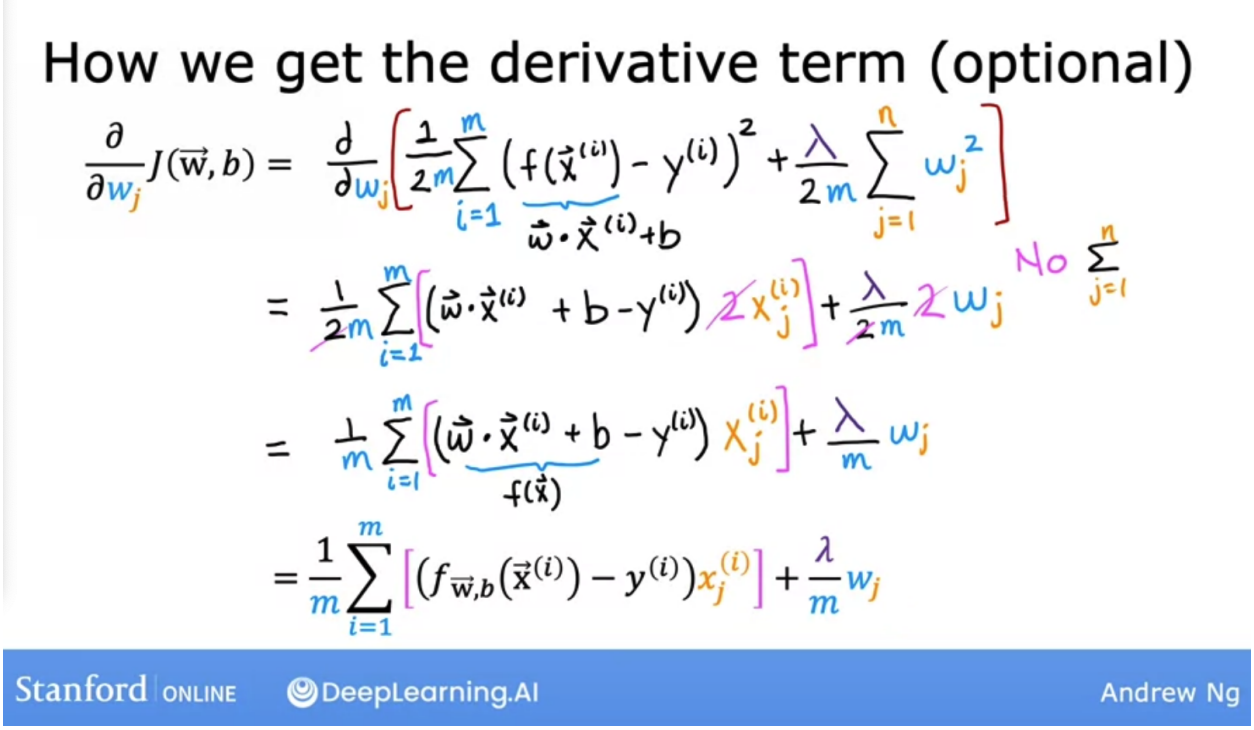
$w_j \left(1 - \alpha \frac{\lambda}{m}\right)$ — usual update

$\alpha \frac{\lambda}{m}$

$0.01 \frac{1}{50} = 0.0002$

$w_j (1 - 0.0002)$

0.9998



— Regularization penalizes “memorization” (over-learning examples).

— Helps the model generalize to unseen examples.

— Changes the representations of learning (either more sparse or more distributed depending on the regularizer).

— Can increase or decrease training time.

— Can decrease training accuracy but increase generalization.

— Works better for large models with multiple hidden layers.

— Generally works better with sufficient data.

▼ How to think about regularization?

— Adds a cost to the complexity of the solution.

— Forces the solution to be smooth.

— Prevents the model from learning item-specific details.

SupervisedMachineLearning-WEEK-3-QUIZ-4

4.4. Types

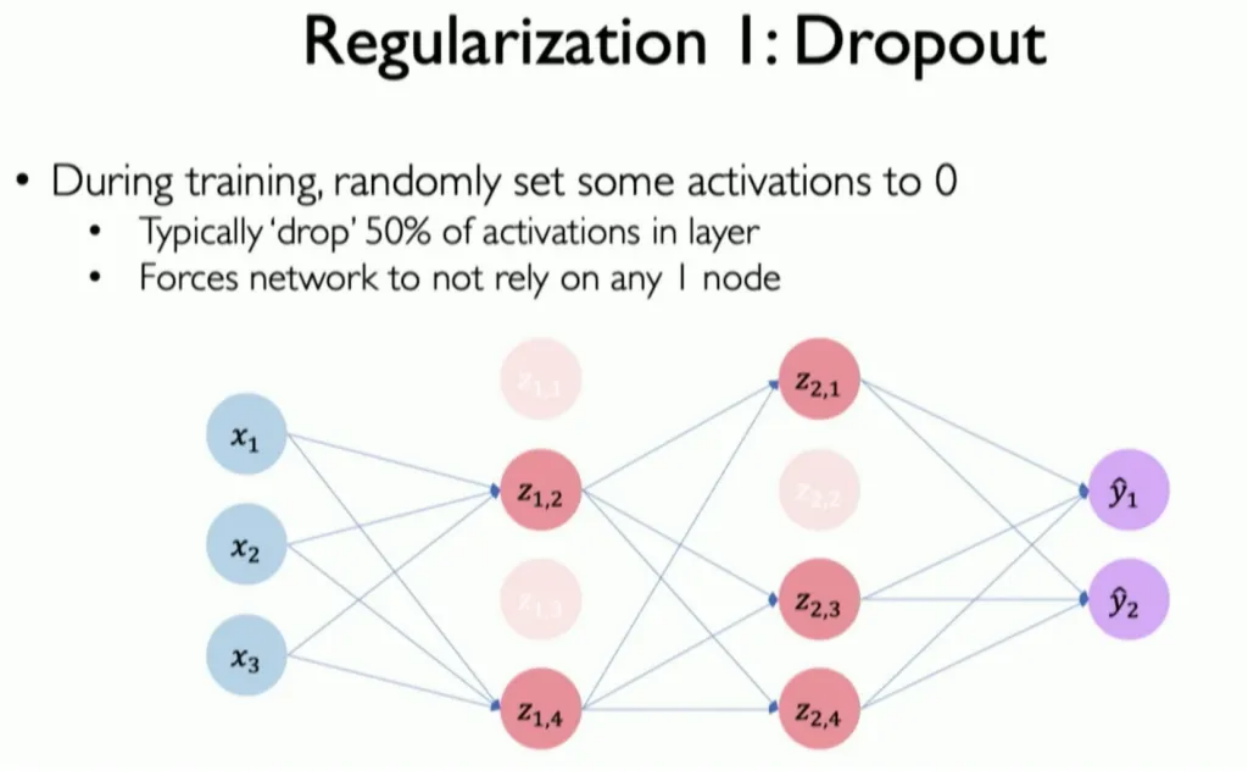
— Modify the model (dropout) → “Node regularization”

— Add a cost to the loss function (L1/2) → “Loss regularization”

— Modify or add data (batch training, data augmentation) → “Data regularization”

4.4.1.1. Dropout

“Remove” nodes randomly during learning (force activation = 0).



▼ EFFECT?

— Prevents a single node from learning too much.

— Forces the model to have distributed representations.

— Makes the model less reliant on individual nodes and thus more stable.

— Generally requires more training (though each epoch computes faster).

— Can decrease training accuracy but increase generalization.

— Usually works better on deep than shallow networks.

— Debate about applying it to convolution layers.

— Works better with sufficient data, unnecessary with “enough” data.

4.4.1.2. L1/L2 regularization

“weight decay” add a cost to the loss function to prevent weights from getting too large.

Loss function without regularization

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}^{(i)}\right)$$

Loss function with regularization

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}^{(i)}\right) + \lambda\sum_{j=1}^{K}||\theta_j^2||$$

Penalty term

— L1 regularization (”lasso”) creates sparse weights by setting some to 0. (sparse weights (lots of 0’s))

— L2 regularization (”ridge” or “weight decay”) shrinks all weights, especially large weights. (>0 in same range)

▼ WHEN TO REGULARIZE?

Regularize as little as possible and as much as necessary.

▼ Other possibilities:

— L1 + L2 (”elastic net” regression)

— Norm of weight matrix

— Sample-specific (positive bias on cancer diagnosis)

▼ Why does regularization reduce overfitting?

— Discourages complex and sample-specific representations.

— Prevents overfitting to training examples.

— Large weights lead to instability (very different outputs for similar inputs).

— In large, complex models with lots of weights (high risk of overfitting).

— Use L1 when trying to understand the important encoding features (more common in regression than DL).

— When training accuracy is much higher than validation accuracy.

4.4.1.3. Data augmentation

“Data augmentation” add more data as slightly modified versions of existing data (usually just for images).

▼ Which regularization method to use?

Oftentimes the “best” method is problem- or architecture-specific.

In many cases, different regularization methods work equally well.

4.5. Training vs. evaluation mode

Gradients are computed only during backprop, not during evaluation.

Some regularization methods are applied only during training, not during evaluation.

Ergo: We need a way to deactivate gradient computations and regularization while evaluating model performance.

4.6. Training in mini-batches

— Batch size is often powers-of-2 (e.g., 2^4 = 16), between 2 and 512.
— Training in batches can decrease computation time because of vectorization (matrix multiplication instead of for-loops).

— But batching can increase computation time for large batches and large data samples (e.g., images).

— Batching is a form of regularization: It smooths learning by averaging the loss over many samples, and thereby reduces overfitting.

https://github.com/SamedyHUNX/supervised-learning