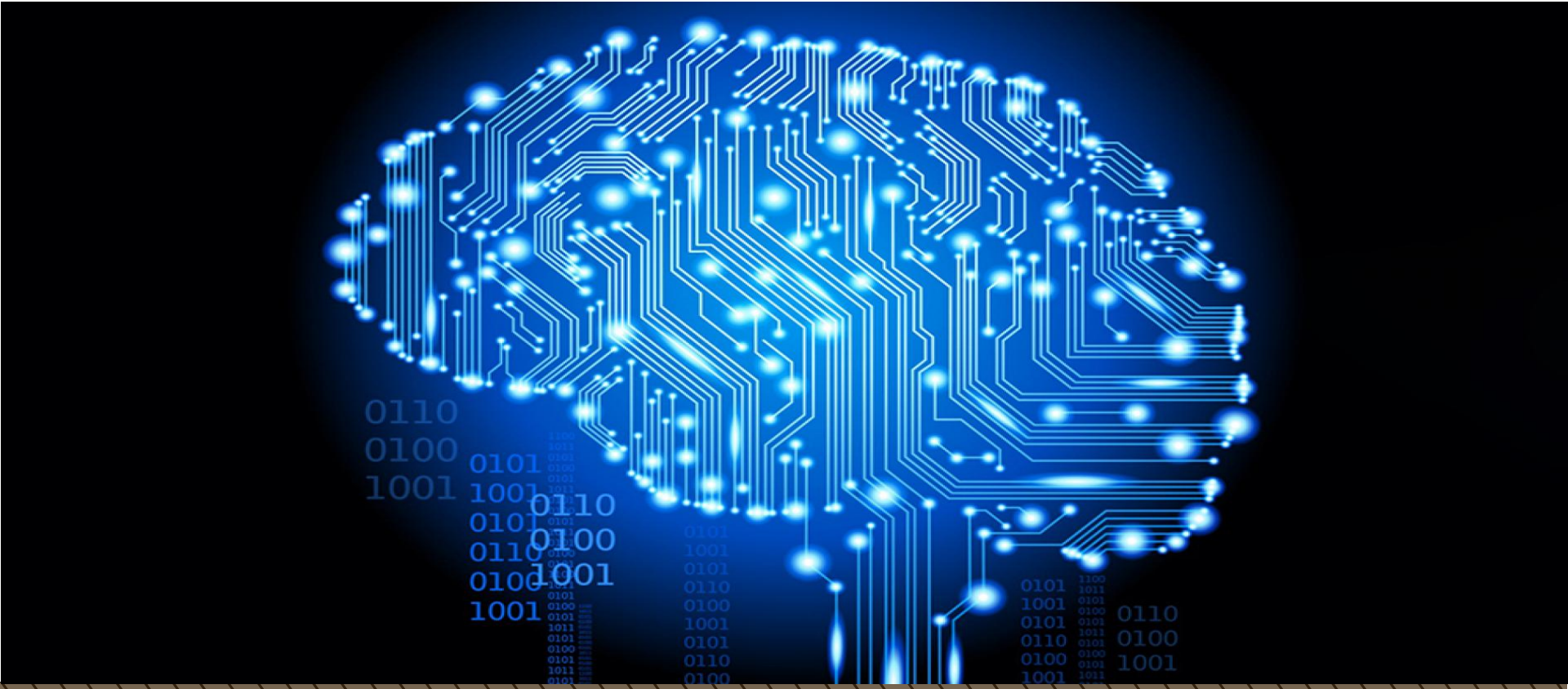


Machine Learning

Lecture 2: Linear Regression

Lecture 2

Linear Regression

Assignment 1

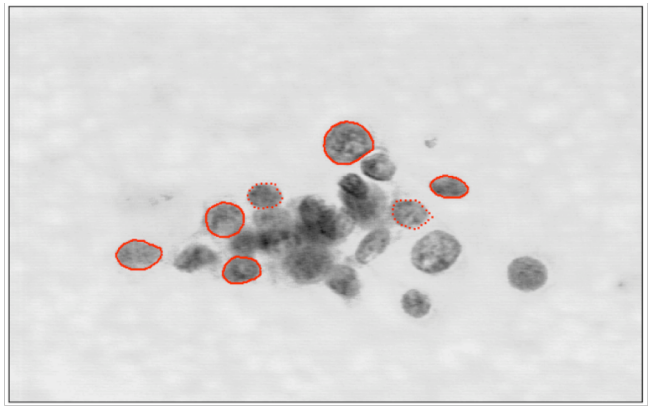
Linear Regression

Terminologies

- Regression
- Regression v.s. Classification

Linear Regression

Terminologies



Images of cell samples were taken from tumors in patients before surgery.
Tumors were then excised through surgery.
Patients were followed to check if the cancer recurred, and how many years they have maintained healthy after the cancer removed.

tumor size	texture	perimeter	···	outcome	time
10.12	33.4	130.1		N	50
11.43	10	101.4		N	43
20.44	15.4	190		Y	15
10.12	33.4	130.1		N	50
11.43	10	101.4		N	43
···					

Linear Regression

Terminologies

tumor size	texture	perimeter	···	outcome	time
10.12	33.4	130.1		N	50
11.43	10	101.4		N	43
20.44	15.4	190		Y	15
10.12	33.4	130.1		N	50
11.43	10	101.4		N	43
···					

- Columns are called input attributes or variables or features.
- The outcome and time are target variables or targets or labels.
- Each row represents an example or instance or sample.
- The whole table is the dataset.

training dataset – a subset of the dataset (i.e., examples) is used to develop a model to predict the targets.

test dataset – once we have our model, another subset of the dataset is used to evaluate the performance of prediction.

Note: These two dataset must be disjoint.

The problem of predicting a discrete/categorical target (e.g., outcome) is called **classification.**

The problem of predicting a continuous valued target (e.g., time) is called **regression.**

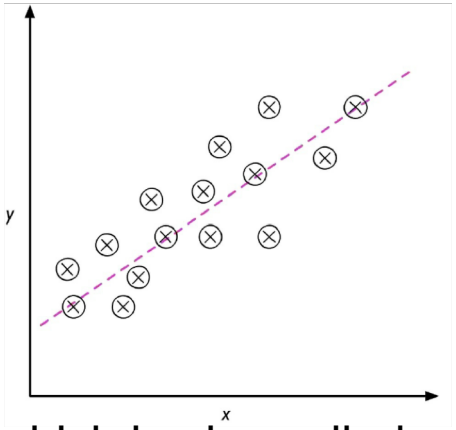
Linear Regression

Terminologies

The term **regression** was devised by Francis Galton in his article Regression towards Mediocrity in Hereditary Stature in 1886.

Galton described the biological phenomenon that the variance of height in a population does not increase over time. He observed that the height of parents is not passed on to their children, but instead the children's height is regressing towards the population mean.

The following figure illustrates the concept of linear regression. Given a predictor variable x and a response variable y, we fit a straight line to this data that minimizes the distance—most commonly the average squared distance—between the sample points and the fitted line. We can now use the intercept and slope learned from this data to predict the outcome variable of new data:



Regression: prediction of **continuous** outcomes, which is also called regression analysis.

Linear Regression

Terminologies

tumor size	texture	perimeter	⋯	⋯	time
$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	⋯	$x_{1,n}$	y_1
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	⋯	$x_{2,n}$	y_2
⋯					
$x_{i,1}$	$x_{i,2}$	$x_{i,3}$	⋯	$x_{4,n}$	y_i
⋯					
$x_{m,1}$	$x_{m,2}$	$x_{m,3}$	⋯	$x_{m,n}$	y_m

How many samples, how many attributes per sample?

- m samples and n attributes per sample.

The sample i contains $\begin{bmatrix} x_{i,1} & x_{i,2} & \cdots & x_{i,n} & y_i \end{bmatrix}$

But we will denote vector $\mathbf{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \cdots \\ x_{i,n} \end{bmatrix} = \vec{x_i} = \underline{x_i}$

We denote the $m \times n$ matrix of attributes by $\mathbf{X}$ and the size-m column vector of outputs from the dataset by $\mathbf{y}$

Linear Regression

Terminologies

- Let X denote the space of input values
- Let Y denote the space of output values.
- Given the dataset $D \subset X \times Y$, find a function:

$$h : X \to Y$$

  such that $h(\mathbf{x})$ is a "good predictor" of the value y.
- h is called hypothesis.
- Supervised learning problems are categorized based on the type of y (i.e., output values):
  - If $Y = \mathbb{R}$, this problem is called regression.
  - If Y is a categorical variable (i.e., discrete set), the problem is called classification.

Linear Regression

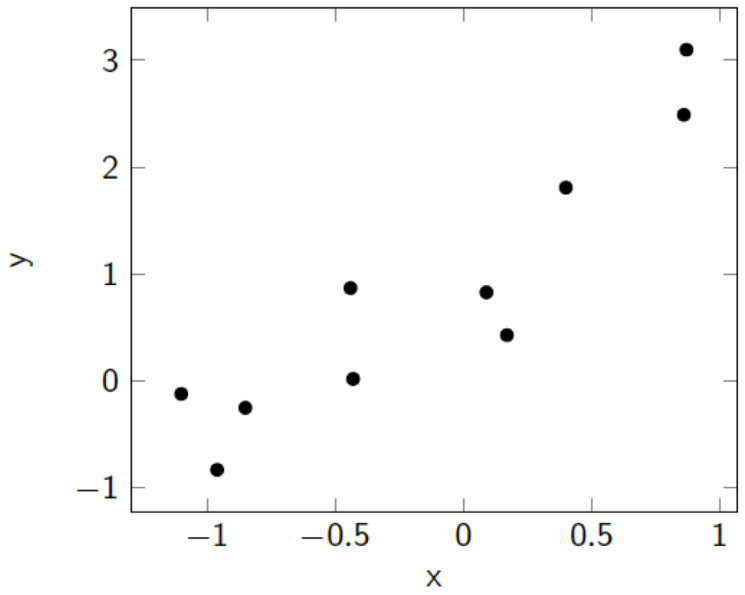
examples

- Consider the following dataset.
- The variable y, `outcome` can take only two values:
  - `Yes`: cancer recurred.
  - `No`: cancer did not recur.
- $h : X \rightarrow Y$
- Question 1: What type of problem is this? (Regression or Classification)?

tumor size	texture	perimeter	$\cdots$	y, outcome
10.12	33.4	130.1		No
11.43	10	101.4		No
20.44	15.4	190		Yes
$\cdots$				

Linear Regression

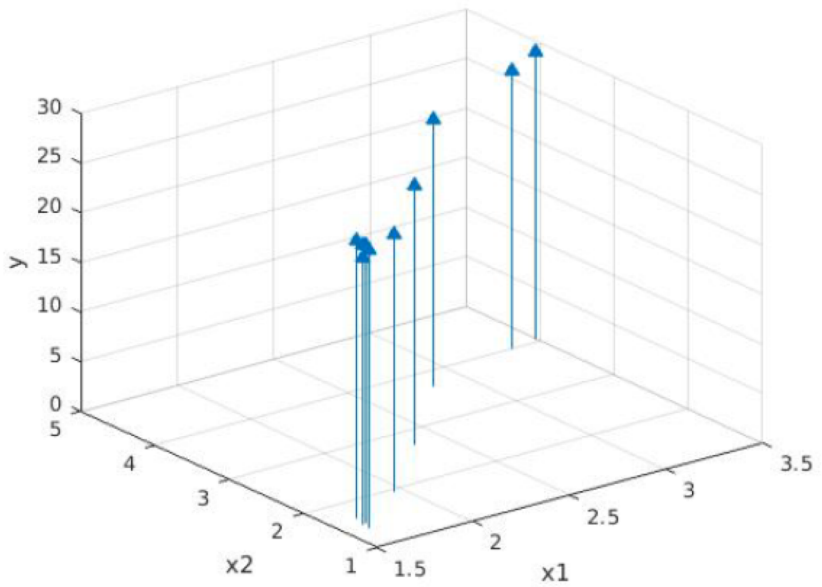
Linear Regression one independent variable



x	y
-1.10	-0.12
-0.96	-0.83
-0.85	-0.25
-0.44	0.87
-0.43	0.02
0.09	0.83
0.17	0.43
0.40	1.81
0.86	2.49
0.87	3.10

Linear Regression

Linear Regression two independent variables



x_1	x_2	y
3.489	4.0375	29
3.354	4.0055	28
2.821	3.6745	27
2.360	2.7205	26
1.948	1.9355	26
1.693	1.6930	27
1.608	1.4805	27
1.618	1.4155	28
1.642	1.6360	28
1.632	1.4940	28

Linear Regression

Linear Regression Linear hypothesis

- Suppose y is a linear function of $\mathbf{x} \in \mathbb{R}^n$:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots w_n x_n$$

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

- w_i of the weight vector $\mathbf{w}$ is called a parameter or weight.
- To simplify notation, we can add an attribute $x_0 = 1$ to the other n attributes:

$$h_{\mathbf{w}}(\mathbf{x}) = \underbrace{w_0 x_0}_{\text{bias term}} + w_1 x_1 + w_2 x_2 + \cdots w_n x_n$$

$$h_w(X) = W\mathrm{X} + \mathrm{b}$$

$$h_w(X) = W\mathrm{X}$$

- Now the hypothesis becomes:

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{n} w_i x_i = \mathbf{w}^T \mathbf{x}$$

where $\mathbf{w}$ and $\mathbf{x}$ are vectors of size $n+1$, i.e., $\underline{w}, \underline{x} \in \mathbb{R}^{n+1}$.

Linear Regression

Linear Regression

$f(X) = W\mathrm{X} + \mathrm{b}$

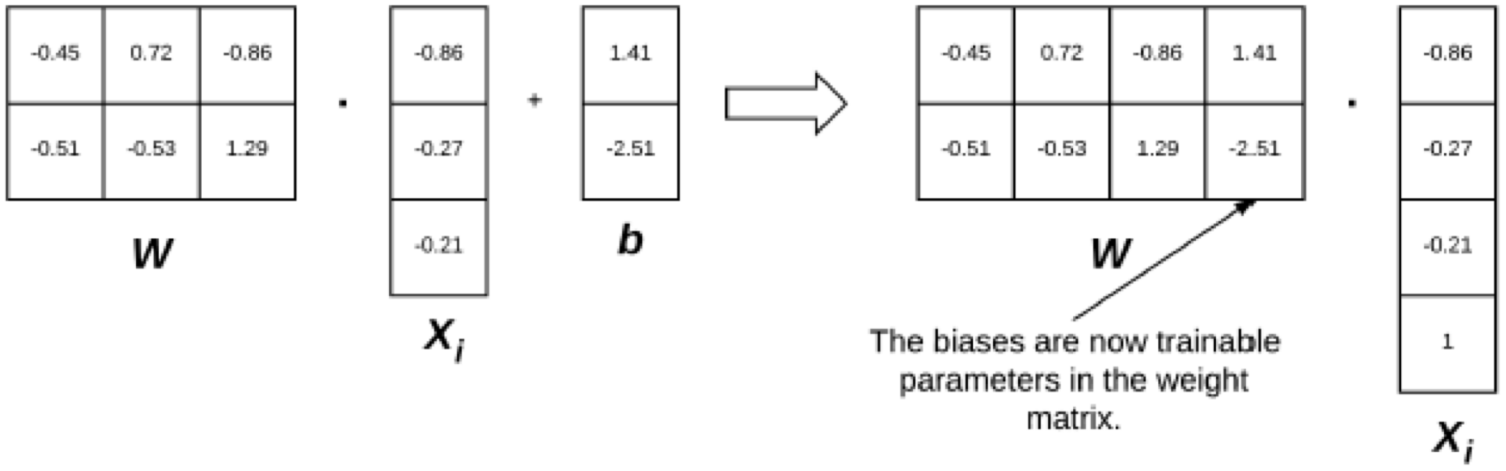
$f(X) = W\mathrm{X}$

$h_w(X) = W\mathrm{X} + \mathrm{b}$

$h_w(X) = W\mathrm{X}$

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

$h_\theta(X) = \theta\ \mathrm{X}$



Linear Regression

Linear Regression Linear hypothesis

- Suppose y is a linear function of $\mathbf{x} \in \mathbb{R}^n$:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots w_n x_n$$

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

- w_i of the weight vector $\mathbf{w}$ is called a parameter or weight.
- To simplify notation, we can add an attribute $x_0 = 1$ to the other n attributes:

$$h_{\mathbf{w}}(\mathbf{x}) = \underbrace{w_0 x_0}_{\text{bias term}} + w_1 x_1 + w_2 x_2 + \cdots w_n x_n$$

- Now the hypothesis becomes:

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{n} w_i x_i = \mathbf{w}^T \mathbf{x}$$

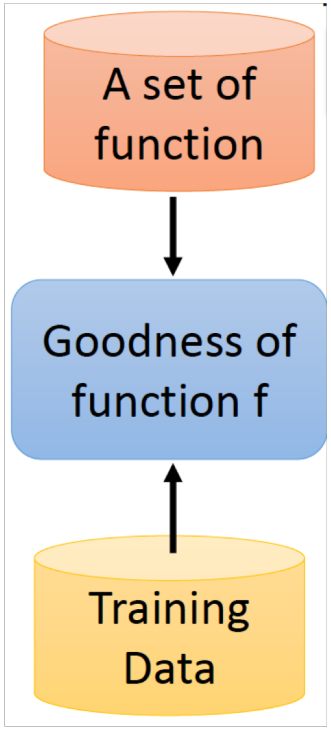
where $\mathbf{w}$ and $\mathbf{x}$ are vectors of size $n+1$, i.e., $\underline{w}, \underline{x} \in \mathbb{R}^{n+1}$.

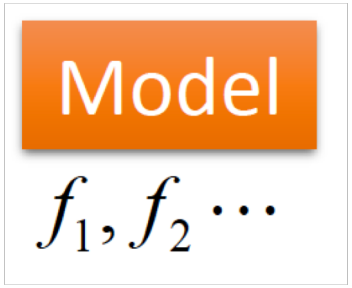
Question: How to solve W?

Linear Regression

Linear Regression

Question: How to solve W?





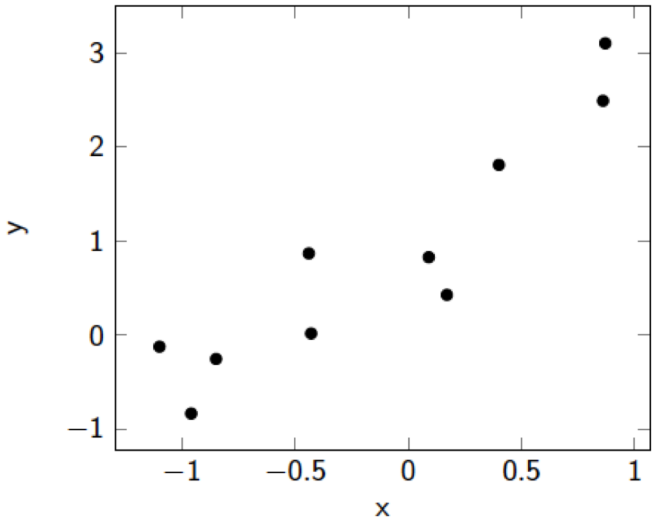
$$y \updownarrow h(x) = b + w \cdot x$$

Step2: goodness of the function

"Loss function" (cost function)
J(w) or L(w)

Linear Regression

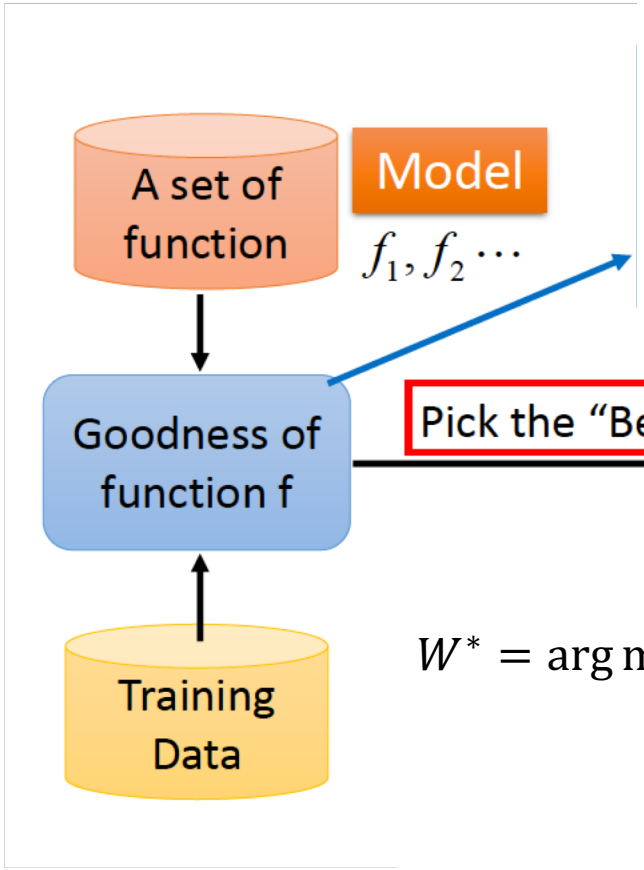
Linear Regression Loss/Cost function



- Main idea: the obtained $\underline{w}$ should make the predictions of h_w close to the true values of $\underline{y}$.
- Define a measure of error function: SSE, Sum of Squared Errors, $J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(h_w(\underline{x_i}) - \underline{y_i}\right)^2$, there are m samples in the training set, and $\underline{x_i}, \underline{w} \in \mathbb{R}^{n+1}$
- Here, $\frac{1}{2}$ factor is there for just convenience in solving for $\underline{w}$.
- Now, the goal is to solve for $\mathbf{w}$ that minimizes $J(\mathbf{w})$, i.e., that minimizes the SSE.

Linear Regression

Linear Regression Loss/Cost function



Loss/Cost function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(h_w(x_i) - y_i\right)^2$$

$$f^* = \arg\min J(w)$$

$$W^* = \arg\min(\,0.5 * \sum_{n=1}^{m}(h(x_i) - y_i)^2)$$

Linear Regression

Linear Regression Loss/Cost function **is a function of W**

Example:
Y= [1, 2, 3]
X= [1, 2, 3]

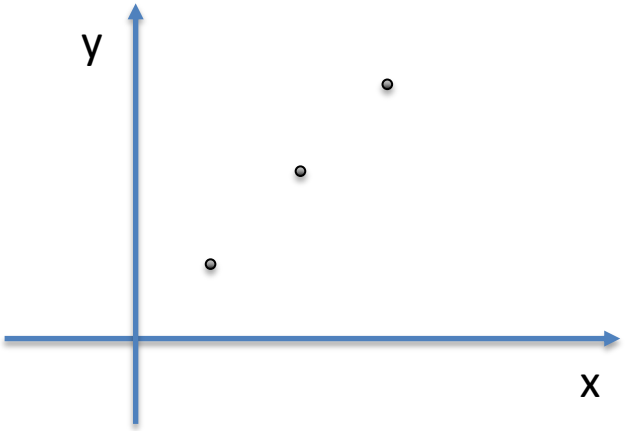
Try H(x) assume that b=0

1) h(x)= 0*x

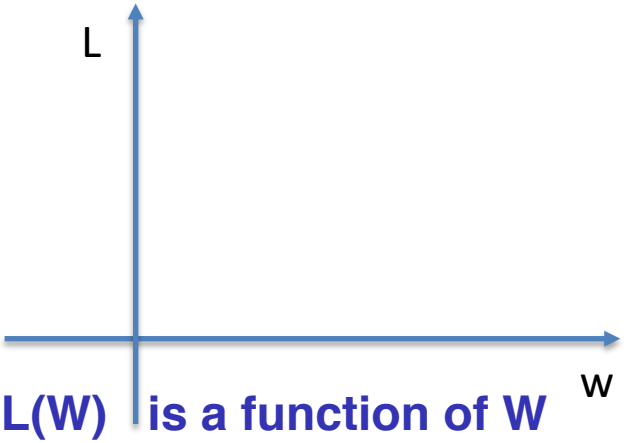
1) h(x)= 0.5*x

2) h(x)=1*x
3) ….

4) h(x) =2*x



h(x) is a function of x



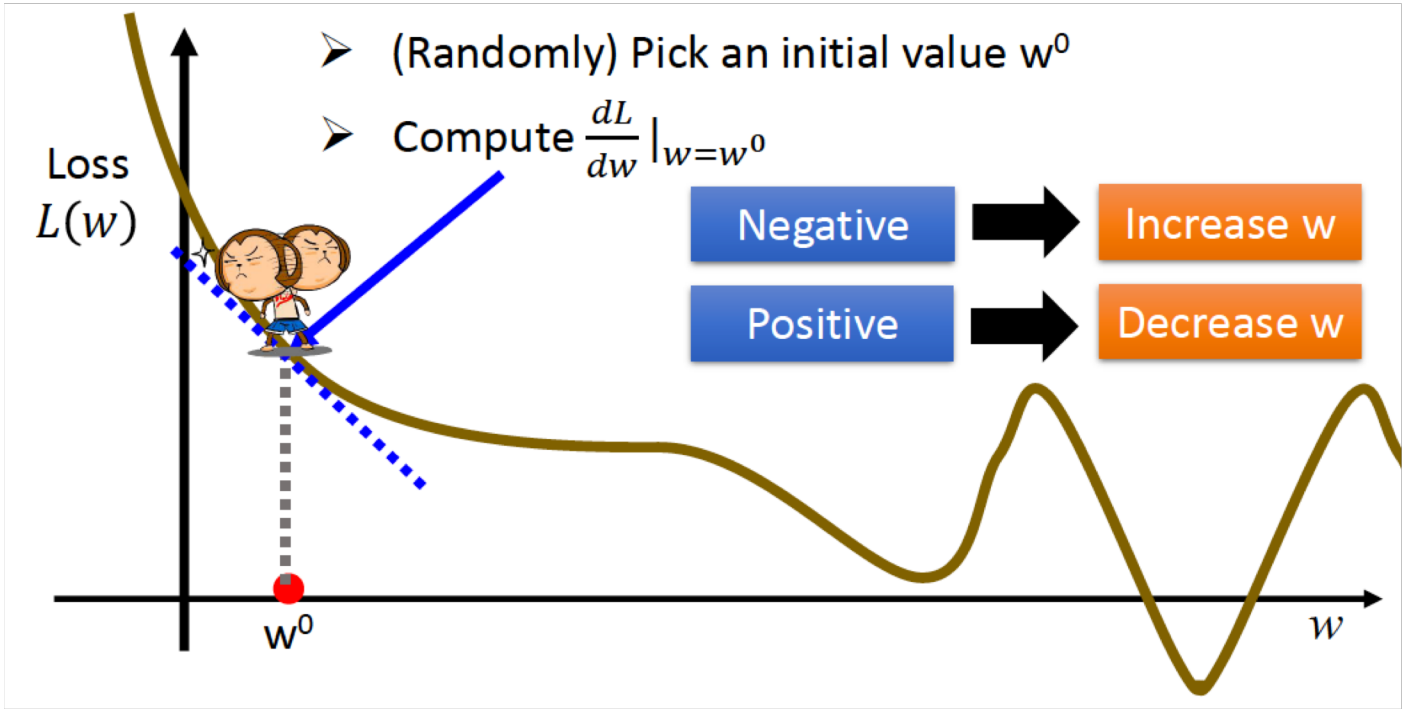
L(W) is a function of W

Linear Regression

Linear Regression Loss/Cost function

Consider loss function $L(w)$ with one parameter w:

$$w* = arg \min L\,(\mathrm{w})$$



Gradient Descent

Introduction

Question: “If we want to reach a global minimum, why not just directly jump to it? It’s clearly visible on the plot?”

Gradient Descent

Introduction

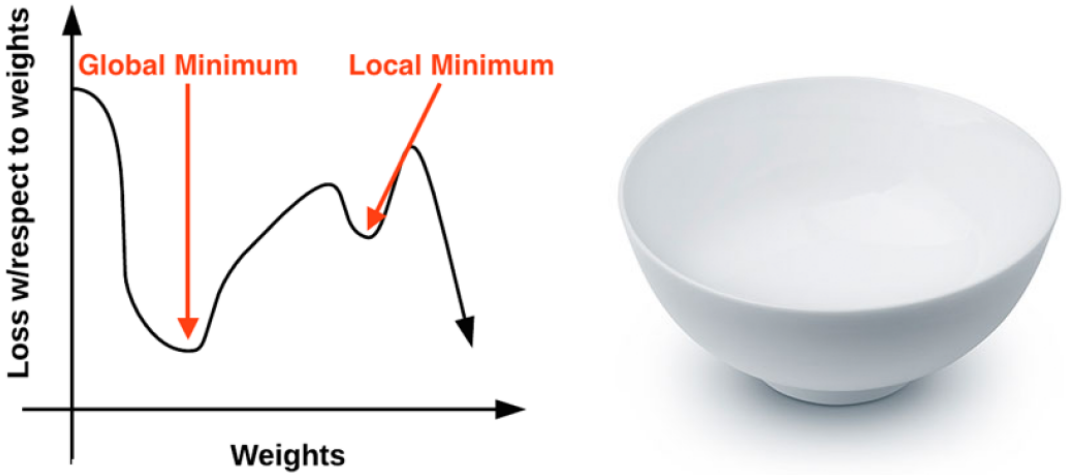
Question: “If we want to reach a global minimum, why not just directly jump to it? It’s clearly visible on the plot?”

- We would be blindly placed somewhere on the plot, having no idea what the landscape in front of us looks like, and we would have to navigate our way to a loss minimum without accidentally climbing to the top of a local maximum.

Gradient Descent

Introduction

- Instead of relying on pure randomness, we need to define an *optimization algorithm* that allows us to literally improve **W** and **b**
- The **gradient descent** method is an *iterative optimization algorithm* that operates over a **loss landscape** (also called an optimization surface).



Left: The "naive loss" visualized as a 2D plot. **Right**: A more realistic loss landscape can be visualized as a bowl that exists in multiple dimensions. Our goal is to apply gradient descent to navigate to the bottom of this bowl (where there is low loss).

Gradient Descent

Introduction

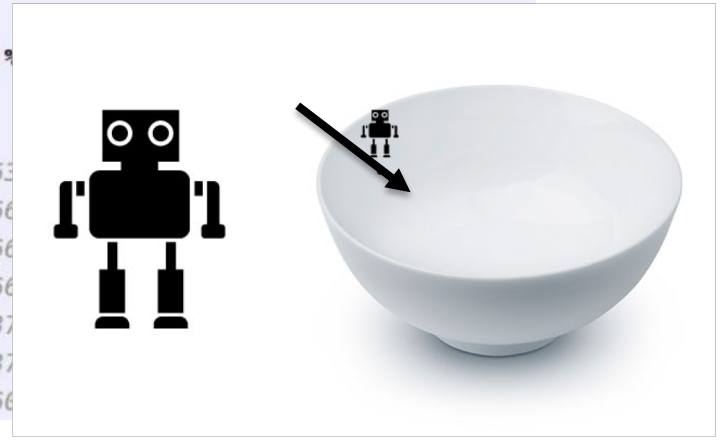
http://cs231n.github.io/optimization-1/

Method 1: Randomly initialize **W** and **b**, evaluate, and repeat over and over again, hoping that at some point we land on a set of parameters that obtains best fit?

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' %

# prints:
# in attempt 0 the loss was 9.401632, best 9.401
# in attempt 1 the loss was 8.959668, best 8.959
# in attempt 2 the loss was 9.044034, best 8.959
# in attempt 3 the loss was 9.278948, best 8.959
# in attempt 4 the loss was 8.857370, best 8.857
# in attempt 5 the loss was 8.943151, best 8.857
# in attempt 6 the loss was 8.605604, best 8.605
# ... (trunctated: continues
```

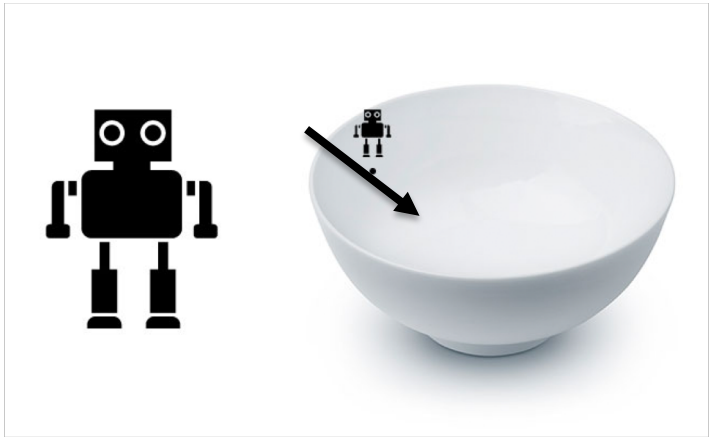


Where to go????? **Direction! Gradient!**

Gradient Descent

Numeric Gradient

Analytic Gradient



Where to go?????

Direction! Gradient!

Gradient Descent

Numeric Gradient

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Problems ?

1) Approximation (Lim of h)
2) Painfully slow!

Gradient Descent

Analytic Gradient

http://cs231n.github.io/optimization-1/

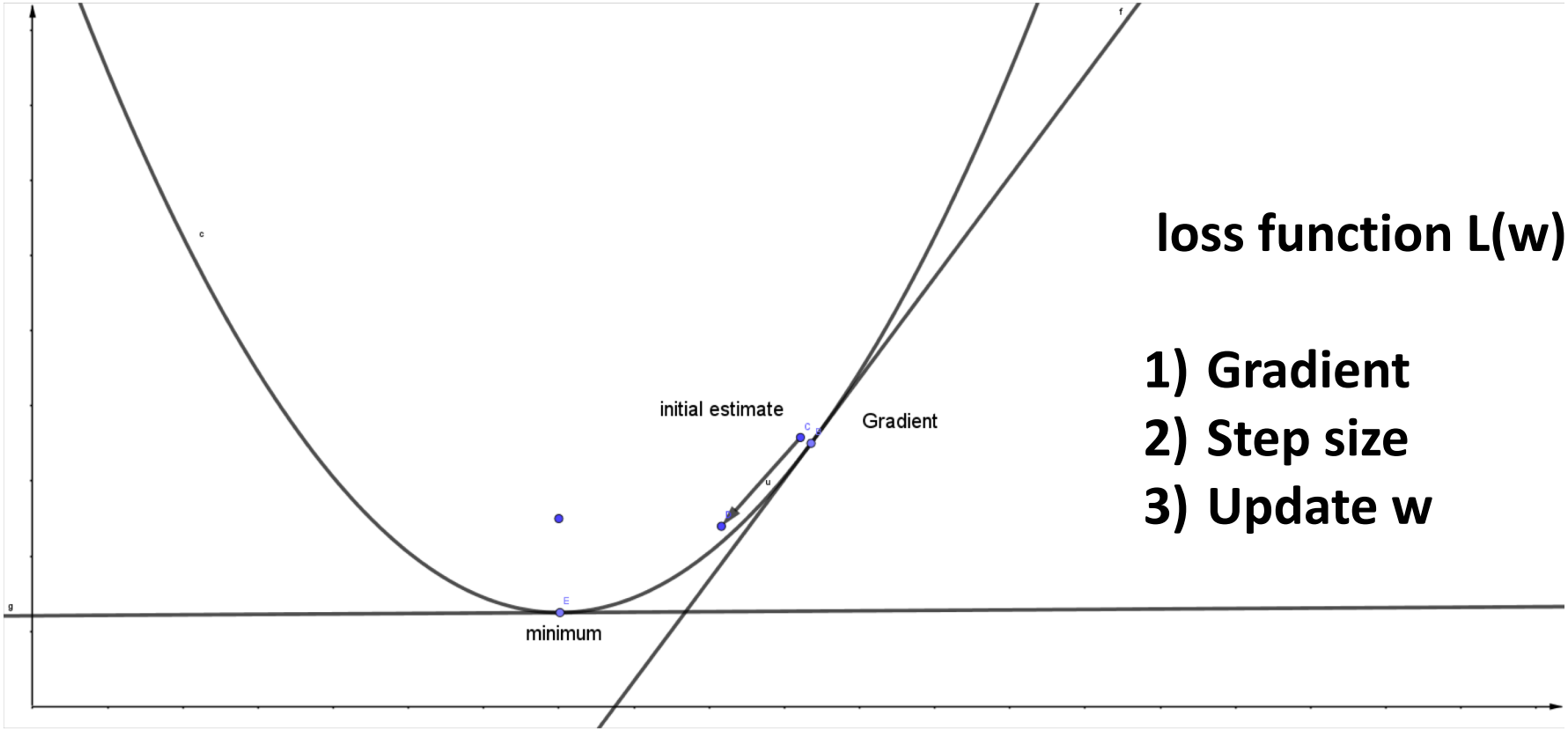
The second way to compute the gradient is analytically using Calculus, which allows us to **derive a direct formula for the gradient** (no approximations) that is also very fast to compute. However, unlike the numerical gradient it can be more error prone to implement, which is why in practice it is very common to compute the analytic gradient and compare it to the numerical gradient to check the correctness of your implementation. This is called a *gradient check.*

e.g. L(w)= w**2 +2w+1

dL/dw =?

Gradient Descent

Gradient Descent (*vanilla gradient descent*)

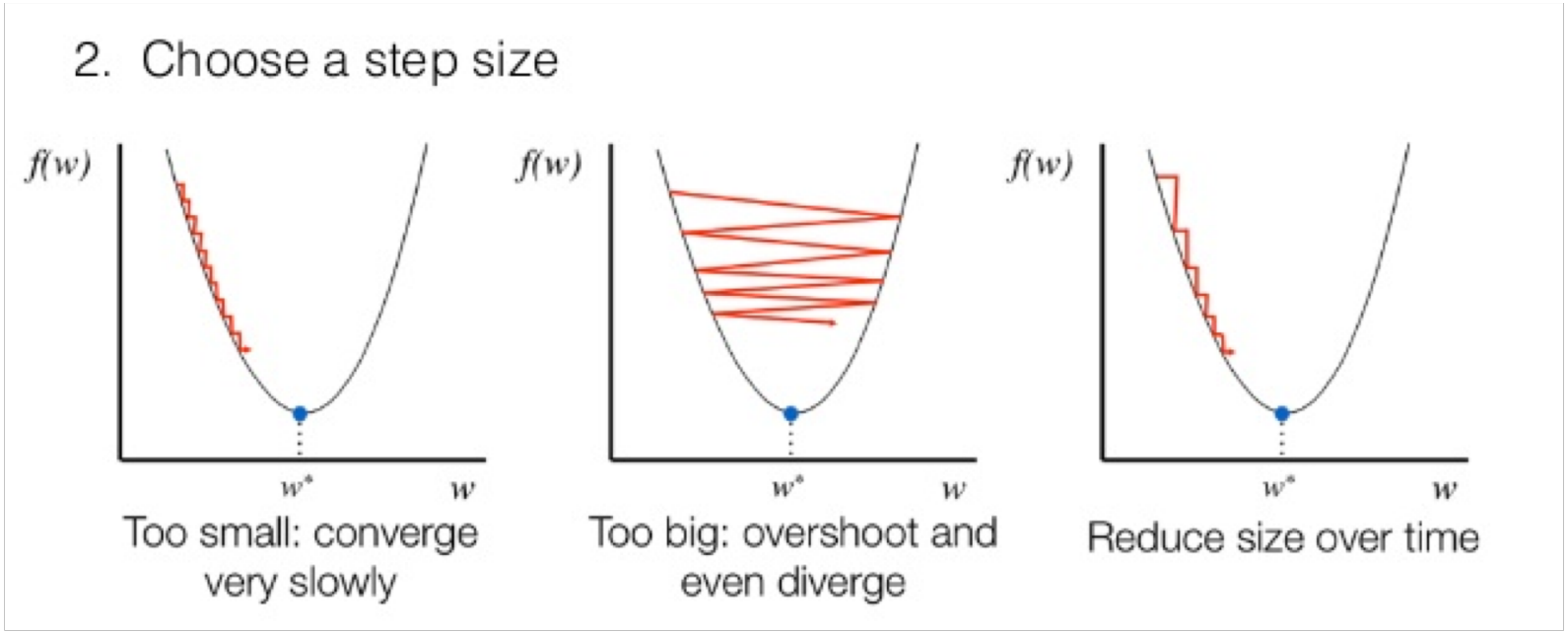


loss function L(w)

1) **Gradient**
2) **Step size**
3) **Update w**

Gradient Descent

Gradient Descent (*vanilla gradient descent*)



Learning rate: α or η

$$w_j := w_j - \alpha \cdot \frac{dL}{dw}$$

0.001 0.0001 0.00001 ....

Gradient Descent

Gradient Descent (*vanilla gradient descent*)

Loss function L(w)

1) **Gradient**
2) **Step size**
3) **Update w**

$$w_j := w_j - \alpha \cdot \frac{dL}{dw}$$

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Gradient Descent

Gradient Descent (*vanilla gradient descent*)

loss function L(w)

1) **Gradient of J or L with W**
2) **Step size**
3) **Update w**

$$\frac{1}{2}\sum_{i=1}^{m}\left(h_w(x_i) - y_i\right)^2$$

$$\frac{dL}{dw} = ?????$$

$$w_j := w_j - \alpha \cdot \frac{dL}{dw}$$

$$w_j := w_j - \alpha \sum_{i=1}^{m}(h_w(x^i) - y^i)x^i$$

$$or: w_j := w_j + \alpha \sum_{i=1}^{m}(y^i - h_w(x^i))x^i$$

The slope(gradient)is the **dot product between the "error" and "x" (features).**

Gradient Descent

Gradient Descent (*vanilla gradient descent*)

loss function L(w)

1) **Gradient of J or L with W**
2) **Step size**
3) **Update w**

$$\frac{1}{2}\sum_{i=1}^{m}\left(h_w(x_i) - y_i\right)^2$$

$$w_j := w_j - \alpha \cdot \frac{dL}{dw}$$

$$w_j := w_j - \alpha \sum_{i=1}^{m}(h_w(x^i) - y^i)x^i$$

Gradient Descent

More notations

- Consider a function $f(u_1, u_2, \cdots, u_n) : \mathbb{R}^n \to \mathbb{R}$
- The partial derivative w.r.to u_i is:

$$\frac{\partial}{\partial u_i} f(u_1, u_2, \cdots, u_n) : \mathbb{R}^n \to \mathbb{R}$$

  the partial derivative is the derivative along the u_i axis, keeping all other variables fixed.
- The gradient $\nabla f(u_1, u_2, \cdots, u_n) : \mathbb{R}^n \to \mathbb{R}$ is a function which outputs a vector containing all the partial derivatives w.r.to u_i:

$$\nabla f = \nabla_{\mathbf{u}} f = \left[\frac{\partial f}{\partial u_1}, \frac{\partial f}{\partial u_2}, \cdots, \frac{\partial f}{\partial u_n} \right]$$

Gradient Descent

More notations

$$
\begin{aligned}
\nabla_{\mathbf{w}} J &= \nabla_{\mathbf{w}} \left[\frac{1}{2} \sum_{i=1}^{m} (h_w(\mathbf{x_i}) - \mathbf{y_i})^2 \right] \\
&= \nabla_{\mathbf{w}} \left[\frac{1}{2} \sum_{i=1}^{m} (\mathbf{x_i w} - y_i)^2 \right] \\
&= \nabla_{\mathbf{w}} \left[\frac{1}{2} (Xw - y)^T (Xw - y) \right] \\
&= \nabla_{\mathbf{w}} \mathrm{tr} \left[\frac{1}{2} (Xw - y)^T (Xw - y) \right] \\
&= \nabla_{\mathbf{w}} \frac{1}{2} \mathrm{tr} \left[(Xw - y)^T (Xw - y) \right] \\
&= \nabla_{\mathbf{w}} \frac{1}{2} \mathrm{tr} \left[w^T X^T X w - y^T X w - w^T X^T y + y^T y \right] \\
&= \nabla_{\mathbf{w}} \frac{1}{2} \left[\mathrm{tr}(w^T X^T X w) - \mathrm{tr}(y^T X w) - \mathrm{tr}(w^T X^T y) \right. \\
&\qquad \left. + \mathrm{tr}(y^T y) \right]
\end{aligned}
$$

Gradient Descent

More notations

$$
\begin{aligned}
\nabla_{\mathbf{w}} J &= \nabla_{\mathbf{w}} \frac{1}{2}\Big[\mathrm{tr}(w^T X^T X w) - \mathrm{tr}(y^T X w) - \mathrm{tr}(w^T X^T y) \\
&\qquad\qquad + \mathrm{tr}(y^T y)\Big] \\
&= \frac{1}{2}\Big[\nabla_{\mathbf{w}}\mathrm{tr}(w^T X^T X w) - \nabla_{\mathbf{w}}\mathrm{tr}(y^T X w) \\
&\qquad\qquad - \nabla_{\mathbf{w}}\mathrm{tr}(w^T X^T y) + \nabla_{\mathbf{w}}\mathrm{tr}(y^T y)\Big] \\
&= \frac{1}{2}\Big[2X^T X w - 2X^T y\Big] \\
&= X^T X w - X^T y
\end{aligned}
$$

If we set gradient to be zero, solve w.

$$
\begin{aligned}
X^T X w - X^T y &= 0 \\
X^T X w &= X^T y \\
(X^T X)^{-1} X^T X w &= (X^T X)^{-1} X^T y \\
w &= (X^T X)^{-1} X^T y
\end{aligned}
$$

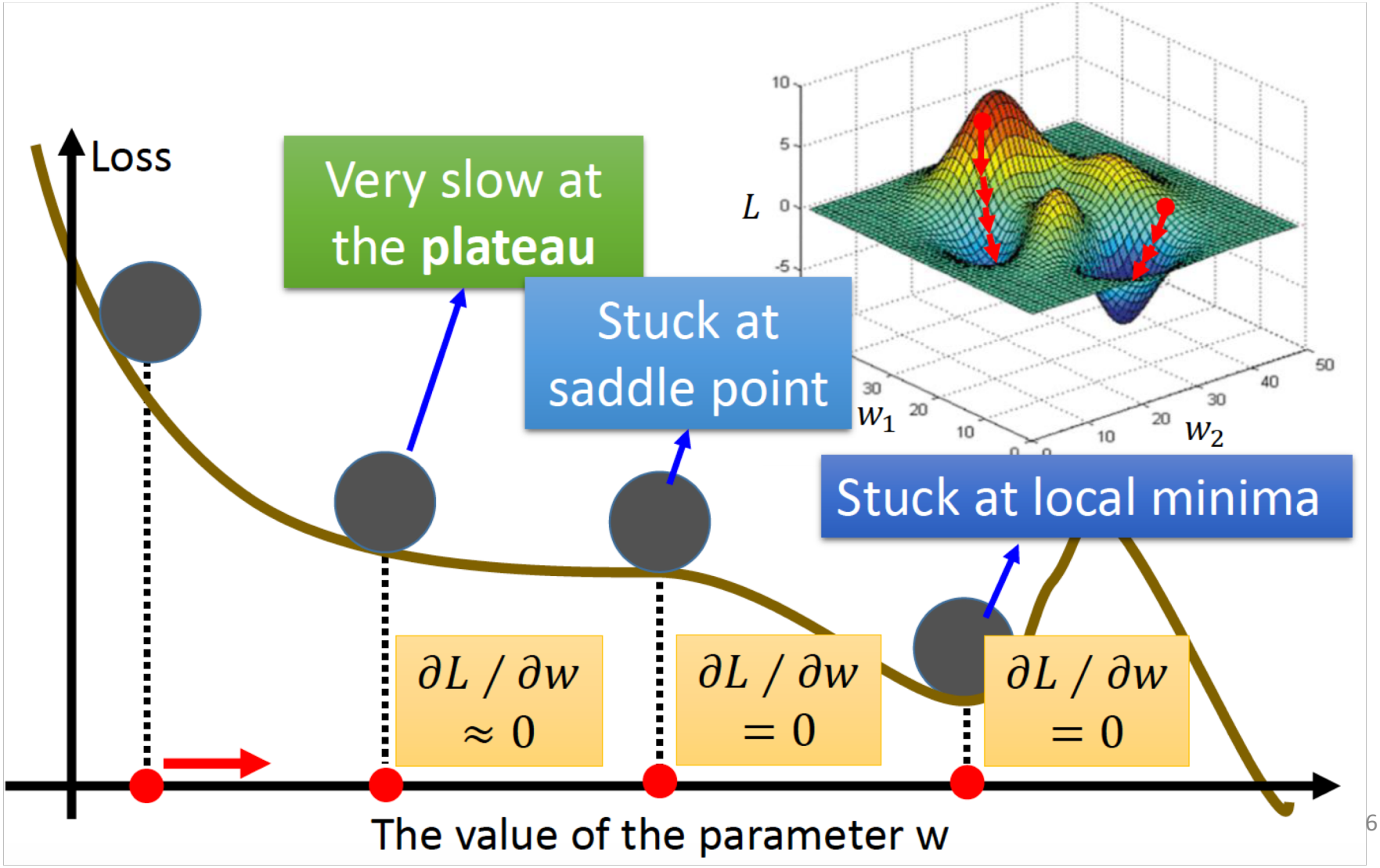
Gradient Descent

More notations

$$
\begin{aligned}
\nabla_{\mathbf{w}} J &= \nabla_{\mathbf{w}} \left[\frac{1}{2} \sum_{i=1}^{m} (h_w(\mathbf{x_i}) - \mathbf{y_i})^2 \right] \\
&= \nabla_{\mathbf{w}} \left[\frac{1}{2} \sum_{i=1}^{m} (\mathbf{x_i w} - y_i)^2 \right] \\
&= \nabla_{\mathbf{w}} \left[\frac{1}{2} (Xw - y)^T (Xw - y) \right] \\
&= \nabla_{\mathbf{w}} \mathrm{tr} \left[\frac{1}{2} (Xw - y)^T (Xw - y) \right] \\
&= \nabla_{\mathbf{w}} \frac{1}{2} \mathrm{tr} \left[(Xw - y)^T (Xw - y) \right] \\
&= \nabla_{\mathbf{w}} \frac{1}{2} \mathrm{tr} \left[w^T X^T X w - y^T X w - w^T X^T y + y^T y \right] \\
&= \nabla_{\mathbf{w}} \frac{1}{2} \left[\mathrm{tr}(w^T X^T X w) - \mathrm{tr}(y^T X w) - \mathrm{tr}(w^T X^T y) \right. \\
&\qquad \left. + \mathrm{tr}(y^T y) \right]
\end{aligned}
$$

Gradient Descent

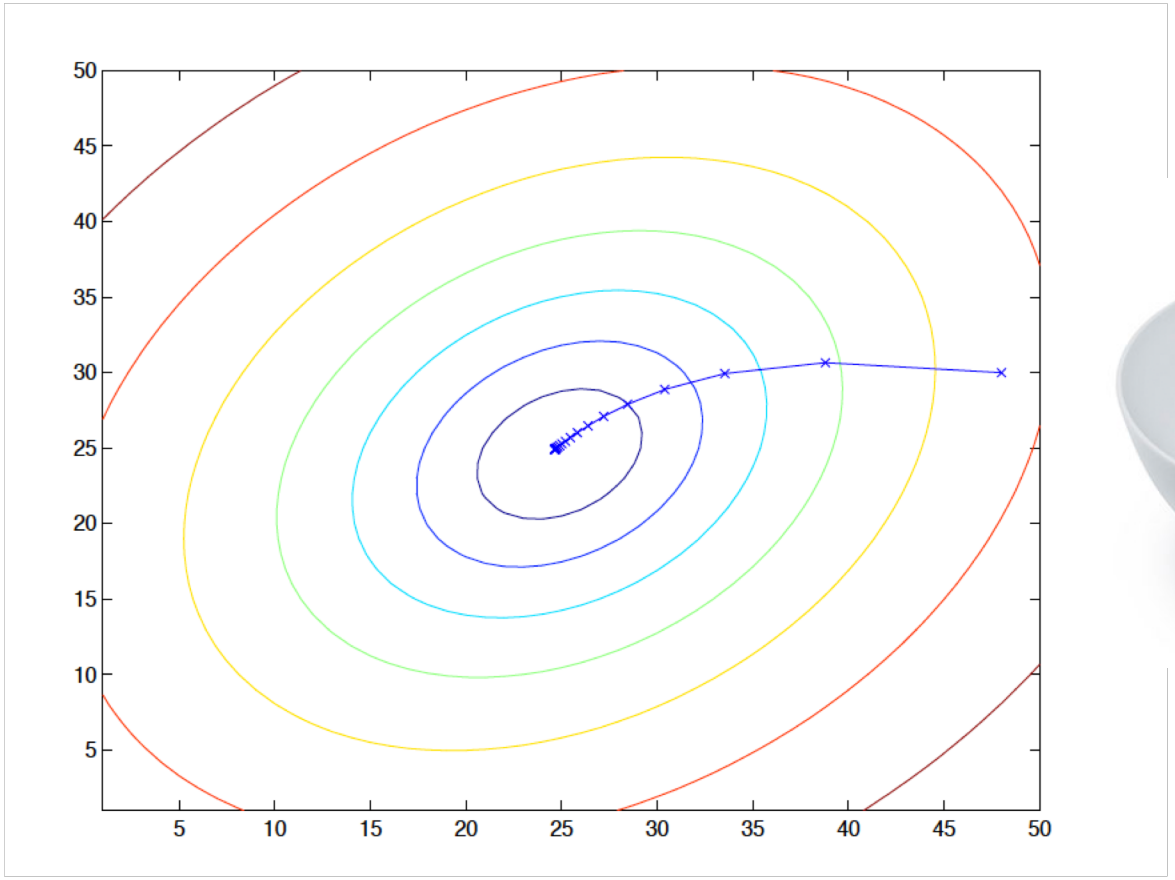
In reality



Gradient Descent

In reality

The loss function L(W) or J(W) is NOT always a convex function.



Gradient Descent

In reality

The loss function L(W) or J(W) is NOT always a convex function.
Book: “Deep learning for computer vision” 9.1.3

9.1.3 Treat It Like a Convex Problem (Even if It’s Not)

Using the a bowl in Figure 9.1 (*right*) as a visualization of the loss landscape also allows us to draw an important conclusion in modern day neural networks – **we are treating the loss landscape as a convex problem, *even if it’s not.*** If some function F is convex, then all local minima are also global minima. This idea fits the visualization of the bowl nicely. Our optimization algorithm simply has to strap on a pair of skis at the top of the bowl, then slowly ride down the gradient until we reach the bottom.

The issue is that nearly all problems we apply neural networks and deep learning algorithms to are *not* neat, convex functions. Instead, inside this bowl we’ll find spike-like peaks, valleys that are more akin to canyons, steep dropoffs, and even slots where loss drops dramatically only to sharply rise again.

Given the non-convex nature of our datasets, why do we apply gradient descent? The answer is simple: *because it does a good enough job*. To quote Goodfellow et al. [10]:

> *“[An] optimization algorithm may not be guaranteed to arrive at even a local minimum in a reasonable amount of time, but it often finds a very low value of the [loss] function quickly enough to be useful.”*

We can set the high expectation of finding a local/global minimum when training a deep learning network, but this expectation rarely aligns with reality. Instead, we end up finding a region of low loss – *this area may not even be a local minimum*, but in practice, it turns out that this is **good enough**.

Gradient Descent

Assignment 1
Solve linear Regression Problem by Gradient Descent
(*vanilla gradient descent*)

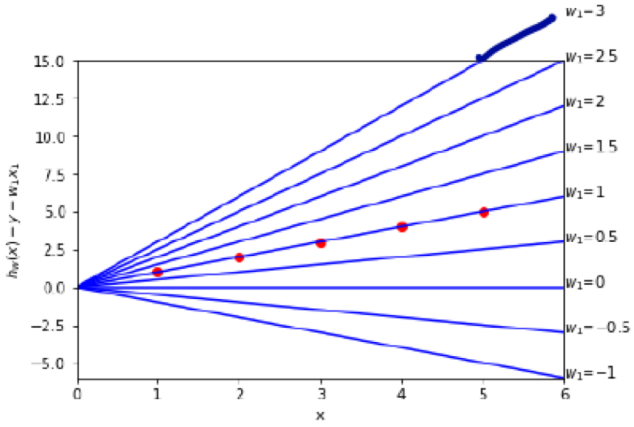
Gradient Descent

Programming

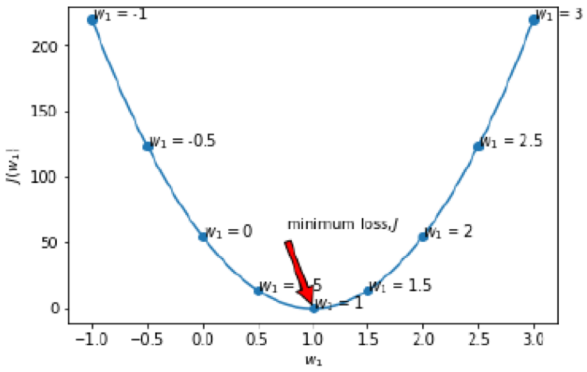
$$\underset{w_0, w_1}{\text{minimize}} \quad J(\underline{w}) = \frac{1}{2} \sum_{i=1}^{m} \left(h_{\mathbf{w}}(x^{\{i\}}) - y^{\{i\}} \right)^2$$

	x	y
0	1	1
1	2	2
2	3	3
3	4	4
4	5	5

Dataset



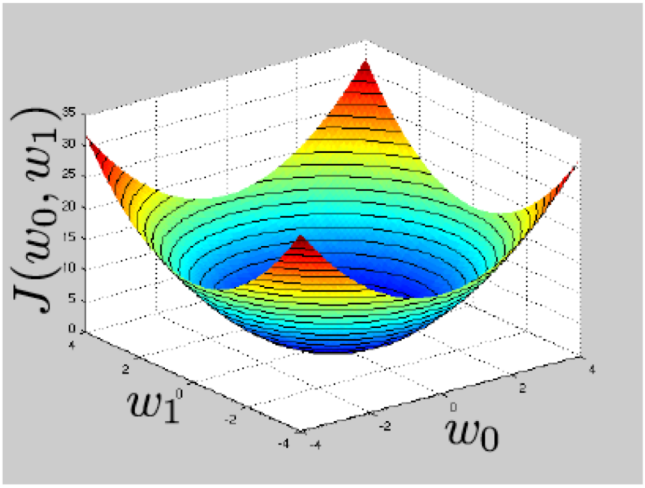
Several regression lines



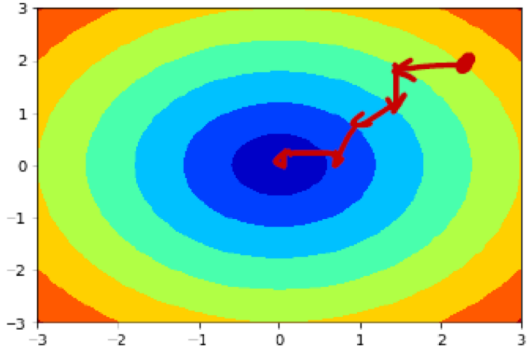
Loss plot

Gradient Descent

Programming



Loss plot



Contour plot of the 3D loss-plot

If the slope >0, we need to decrease w1
w1 = w1 - lr* slope (a negative number, slope)

If the slope < 0, we need to increase w1
w1 = w1 - lr* slope (a negative number, slope)

Gradient Descent

Programming

initialization with random $\mathbf{w} = [w_0, w_1 \cdots w_n]^T$;
repeat
    $w_j \leftarrow w_j - \alpha \cdot \frac{\partial}{\partial w_j} J(\mathbf{w})$;
    (for all n variables)
until *convergence*;

If the slope >0, we need to decrease w1
w1 = w1 - lr* slope (a negative number, slope)

If the slope < 0, we need to increase w1
w1 = w1 - lr* slope (a negative number, slope)

Gradient Descent

Programming

$$
\begin{aligned}
h_{\mathbf{w}}(x) &= w_0 x_0 + w_1 x_1 \\
J(w_0, w_1) &= \frac{1}{2} \sum_{i=1}^{m} \left(h_{\mathbf{w}}(x^{\{i\}}) - y^{\{i\}} \right)^2 \\
&= \frac{1}{2} \sum_{i=1}^{m} \left(w_0 x_0^{\{i\}} + w_1 x_1^{\{i\}} - y^{\{i\}} \right)^2
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial w_0} J(w_0, w_1) &= \sum_{i=1}^{m} \left(h_{\mathbf{w}}(x^{\{i\}}) - y^{\{i\}} \right) \\
\frac{\partial}{\partial w_1} J(w_0, w_1) &= \sum_{i=1}^{m} \left(h_{\mathbf{w}}(x^{\{i\}}) - y^{\{i\}} \right) \cdot x_1^{\{i\}}
\end{aligned}
$$

Gradient Descent

Programming

initialization with random $\mathbf{w} = [w_0, w_1 \cdots w_n]^T$;
repeat
  $w_j \leftarrow w_j - \alpha \cdot \frac{\partial}{\partial w_j} J(\mathbf{w})$;
  (for all n variables)
until *convergence*;

initialization with random $\mathbf{w} = [w_0, w_1]^T$;
repeat

$$w_0 = w_0 - \alpha \cdot \sum_{i=1}^{m} \left(h_{\mathbf{w}}(x^{\{i\}}) - y^{\{i\}} \right)$$

$$w_1 = w_1 - \alpha \cdot \sum_{i=1}^{m} \left(h_{\mathbf{w}}(x^{\{i\}}) - y^{\{i\}} \right) \cdot x_1^{\{i\}}$$

until *convergence*;