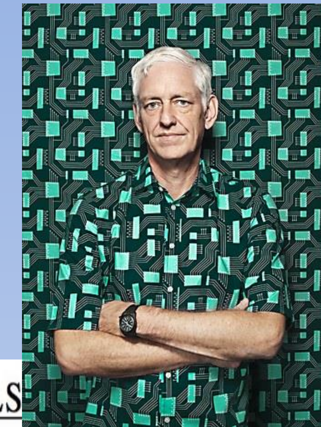
Linear Discrimination

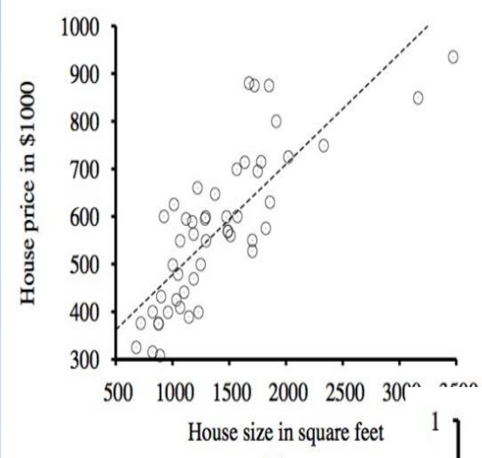
Machine Learning

Chapter 18: Regression & Classification

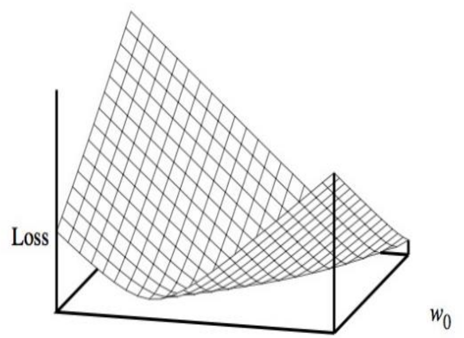


18.6 REGRESSION AND CLASSIFICATION WITH LINEAR MODELS

n decision trees and lists to a different hypothesis space, one
of years: the class of **linear functions** of continuous-valued



(a)

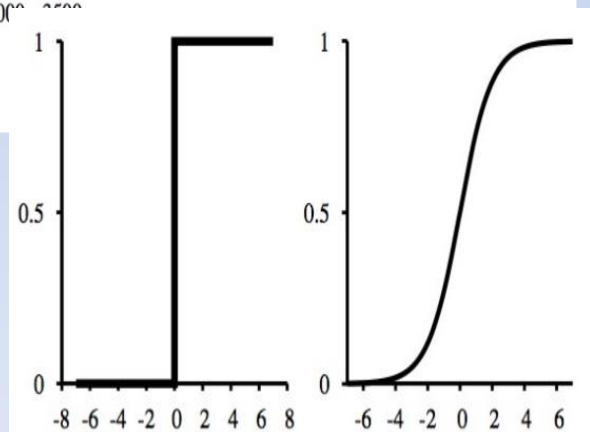


$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$



(a) (b)

Chapter 18.6

- 18.6.1 Linear Regression with 1 input and 1 output
- 18.6.2 Linear Regression with multiple inputs and 1 output
- 18.6.3 Linear Classifiers
- 18.6.4 Logistic Regression

Likelihood- vs. Discriminant-based Classification

- Likelihood-based: Assume a model for $p(\boldsymbol{x}|C_i)$, use Bayes' rule to calculate $P(C_i|\boldsymbol{x})$

  $$g_i(\boldsymbol{x}) = \log P(C_i|\boldsymbol{x})$$

- Discriminant-based: Assume a model for $g_i(\boldsymbol{x}|\Phi_i)$; no density estimation
- Estimating the boundaries is enough; no need to accurately estimate the densities inside the boundaries

Linear Discriminant

- Linear discriminant:

$$g_i(\mathbf{x} \mid \mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0} = \sum_{j=1}^{d} w_{ij} x_j + w_{i0}$$

- Advantages:
  - Simple: O(d) space/computation
  - Knowledge extraction: Weighted sum of attributes; positive/negative weights, magnitudes (credit scoring)
  - Optimal when $p(\mathbf{x} \mid C_i)$ are Gaussian with shared cov matrix; useful when classes are (almost) linearly separable

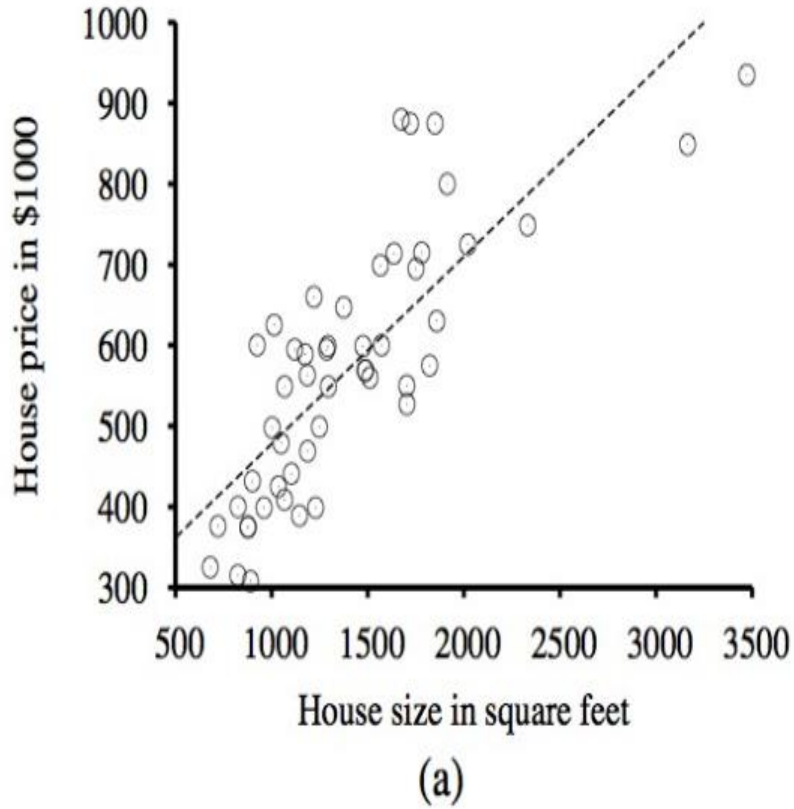
Linear Models

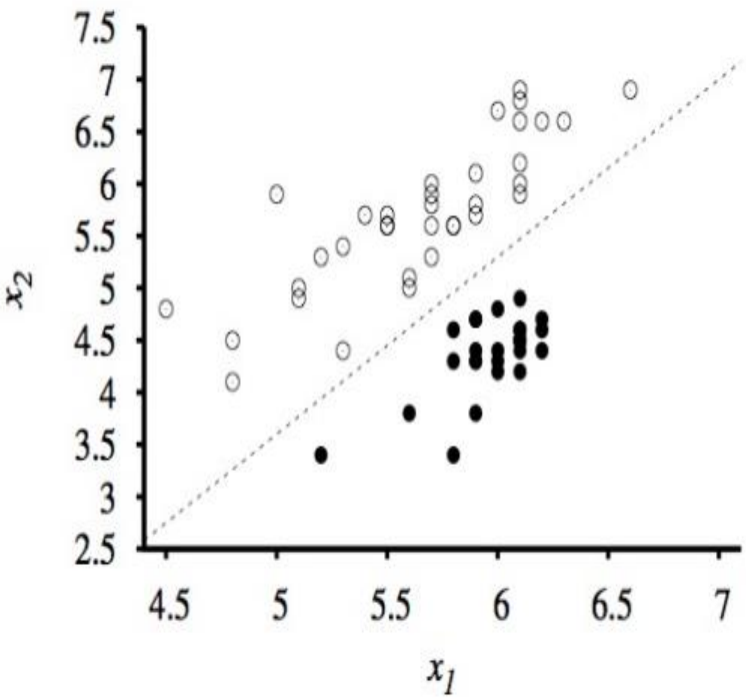
- Linear Regression
- Logistic Regression
- Perceptron
  - Neural Model

- Leads to Multi-Layer Models
  - Neural Nets
  - Deep Learning

Linear Models

- Regression
- Classification





$$g_i(\mathbf{x}|\mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T\mathbf{x} + w_{i0} = \sum_{j=1}^{d} w_{ij}x_j + w_{i0}$$

Linear Models - Historically

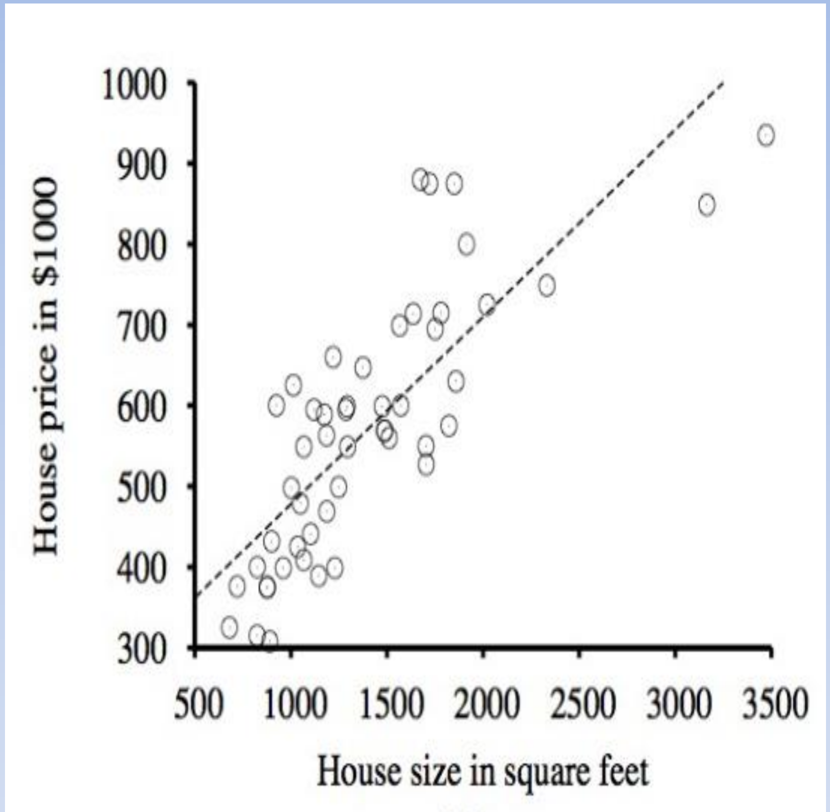
Gauss and Method of Least Squares (1795)

- Carl Friedrich Gauss (1777–1855)
- Uses this method to calculate the orbits of celestial bodies

Evaluate Hypothesis $h_w(x)$

- Define Empirical Loss measured by squared loss function summed over all examples:

$$\sum_{1}^{N}(y_j - h_w(x_j))^2$$



(a)

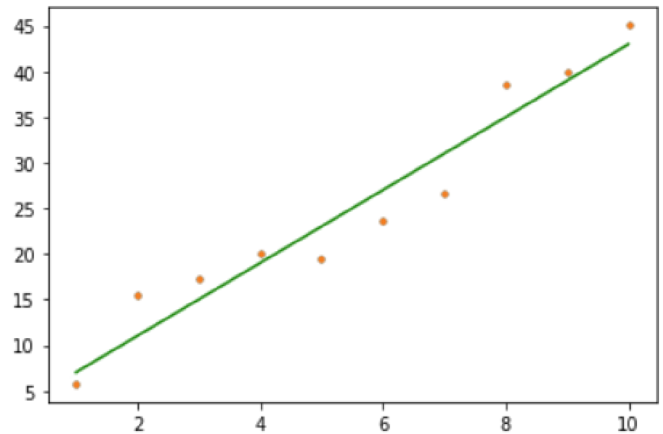
Example
(3, 4) w/ Noise

- (1, 5.75)
- (2, 15.51)
- (3, 17.32)
- (4, 19.99)
- (5, 19.56)
- (6, 23.56)
- (7, 26.58)
- (8, 38.66)
- (9, 40.01)
- (10, 45.08)

[20]

```
Noise = lambda eps: np.random.random()*eps - (eps/2)
F = lambda x: 3 + 4*x
X = list(range(1, 11))
print (X)

Y = [round(F(x) + Noise(10),2) for x in X]

print (Y)

### Add a little noise
plt.plot(X,Y,'.')
Hxs = [F(x) for x in X]

plt.plot(X, Y, '.')
plt.plot(X, Hxs, '-', color="green")
plt.show()
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[5.75, 15.51, 17.32, 19.99, 19.56, 23.56, 26.58, 38.66, 40.01, 45.08]
```



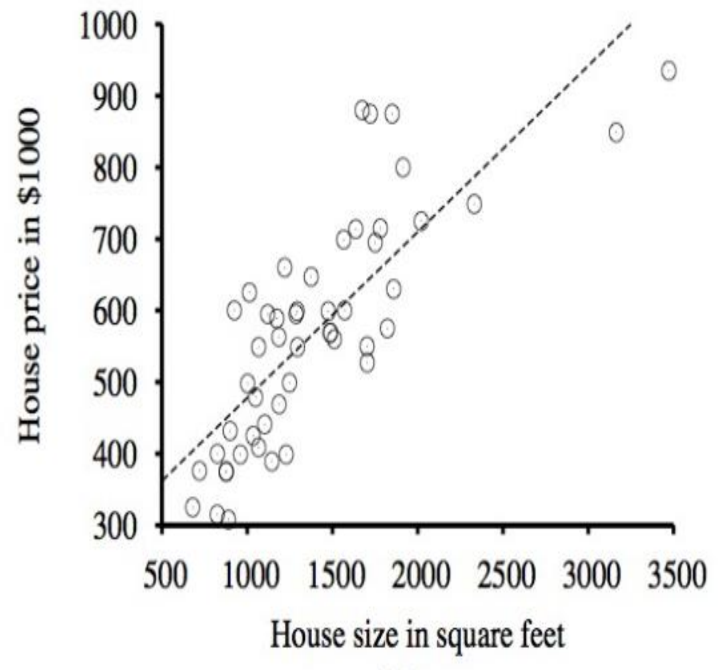
Univariate Linear Regression

- ASSUME: $h_w(x) = w_1 x + w_0$
- Find the value of the weights that minimize empirical loss!
- Loss function becomes:

$$\sum_{1}^{N}(y_j - h_w(x_j))^2$$

$$\sum_{1}^{N}(y_j - (w_1 x_j + w_0))^2$$

- How do we find w_0 & w_1



(a)

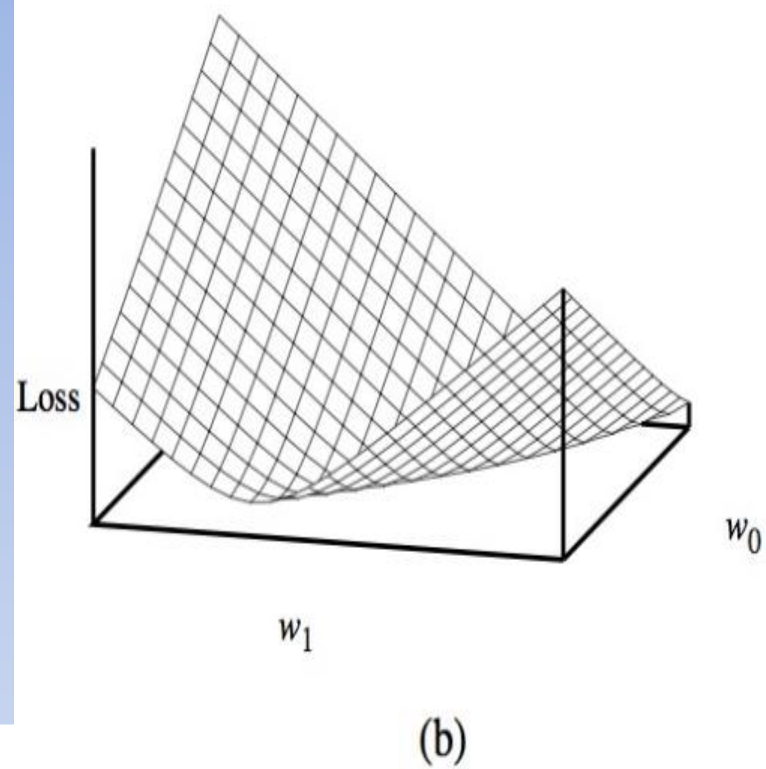
Introduce Calculus
Loss Function Derivative w/ w_0 & w_1

$$\frac{\partial}{\partial w_0}\left(\sum_1^N (y_j - (w_1 x + w_0))^2\right) = 0$$

$$\frac{\partial}{\partial w_1}\left(\sum_1^N (y_j - (w_1 x + w_0))^2\right) = 0$$

- Minimize Loss
- Solve for First Derivative equaling 0

Least Squares Regression



$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \quad w_0 = (\sum y_j - w_1(\sum x_j))/N \,. \qquad (18.3)$$

- Minimizes loss function
- 18.13b (above) visualizes loss function
  - convex

$$\beta = \frac{cov(x,y)}{var(x)}$$

$$\operatorname{cov}(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - E(X))(y_i - E(Y)).$$

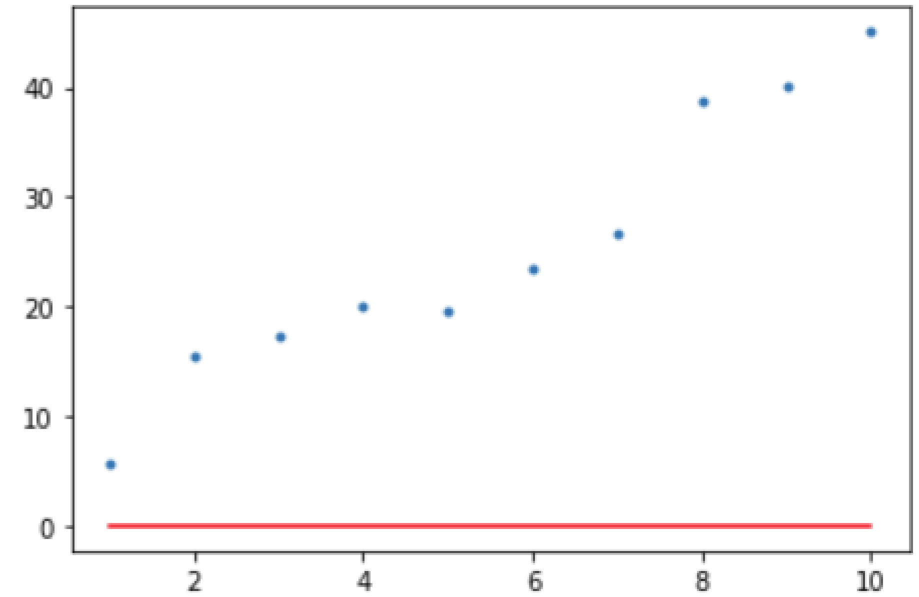
$$\alpha = \bar{y} - \beta\bar{x}$$

Simple Hypothesis

- 0, 0
  - w0 = 0
  - w1 = 0

```
# Plot Hypothesis
Hx = lambda w, x: w[0] + w[1]*x

w = 0, 0

Hxs = [Hx(w,x) for x in X]
print (Hxs)

plt.plot(X, Y, '.')
plt.plot(X, Hxs, '-', color="red")
```

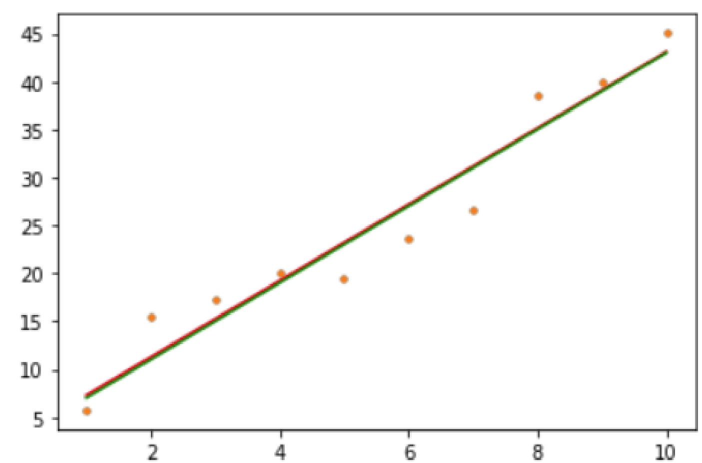
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[<matplotlib.lines.Line2D at 0x7f85f84ebb00>]
```



Least Squares Regression

```python
cv = np.cov(X, Y, bias=True, rowvar=False)
w1 = round(cv[1][0]/cv[0][0],2)
w0 = round(np.mean(Y) - w1*np.mean(X),2)
print (w0, w1)

Hxs = [Hx((w0,w1), x) for x in X]

plt.plot(X, Y, '.')
plt.plot(X, Hxs, '-', color="red")

Hxs = [F(x) for x in X]

plt.plot(X, Y, '.')
plt.plot(X, Hxs, '-', color="green")

plt.show()
```

```
3.31 3.98
```



Another View: Vectorized

$$Y = X\beta$$

- We know X
- We know Y
- Solve for B.

Solution

$$\beta = (X^T X)^{-1} X^T Y$$

- # In[1]:

  from numpy.linalg import inv

  from numpy import dot, transpose

```
print(dot(inv(dot(transpose(X), X)), dot(transpose(X), R)))
```

```python
ones = np.ones([len(X),1])
Xv = np.append(ones, np.array(X).reshape(-1,1),1)
print (Xv)
a0 = np.dot(np.transpose(Xv),Xv)
a1 = np.linalg.inv(a0)
b = np.dot(np.transpose(Xv), Y)
print ([round(v, 2) for v in np.dot(a1,b)])
```

```
[[ 1.  1.]
 [ 1.  2.]
 [ 1.  3.]
 [ 1.  4.]
 [ 1.  5.]
 [ 1.  6.]
 [ 1.  7.]
 [ 1.  8.]
 [ 1.  9.]
 [ 1. 10.]]
[3.34, 3.98]
```

Alternatively

- # In[1]:
  from numpy.linalg import lstsq

  print(lstsq(X, R)[0])

```
1 print([round(v,2) for v in np.linalg.lstsq(Xv,Y,rcond=None)[0]])
```

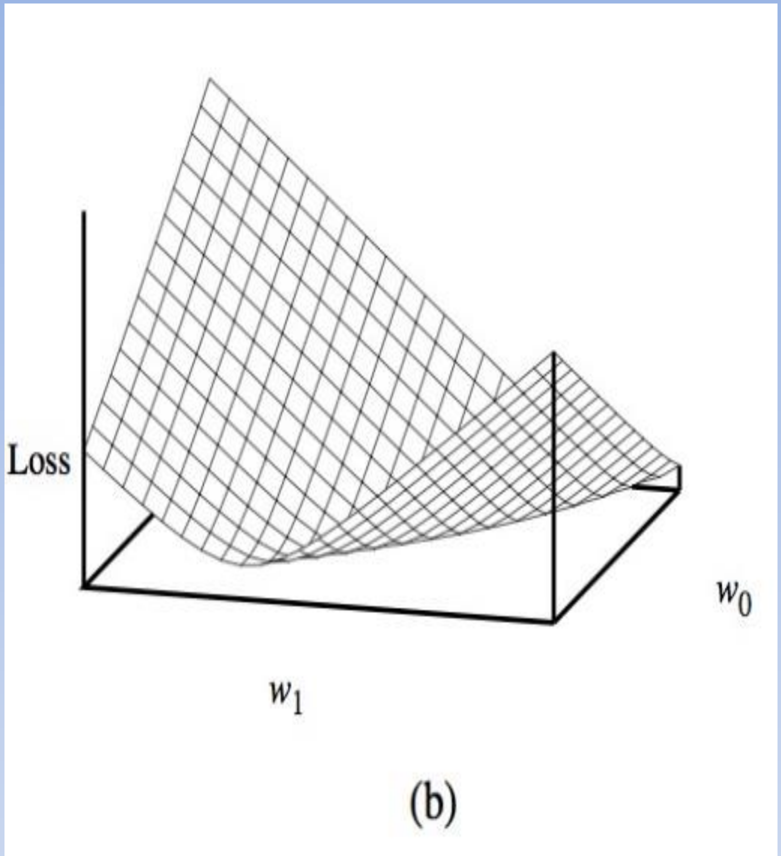
```
[3.34, 3.98]
```

Gradient Descent

- Closed form difficult with additional variables
- Optimizing weights becomes a search problem
- Hill-Climbing w/ Gradient Descent works well!

Gradient Descent

- Choose arbitrary starting location in weight space
- Move to a neighboring point lower in weight space
- Continue to move to neighboring points lower in weight space till converging.



Gradient-Descent

- $E(\boldsymbol{w}|X)$ is error with parameters $\boldsymbol{w}$ on sample X

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w} \mid X)$$

- Gradient

$$\nabla_w E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_d} \right]^T$$

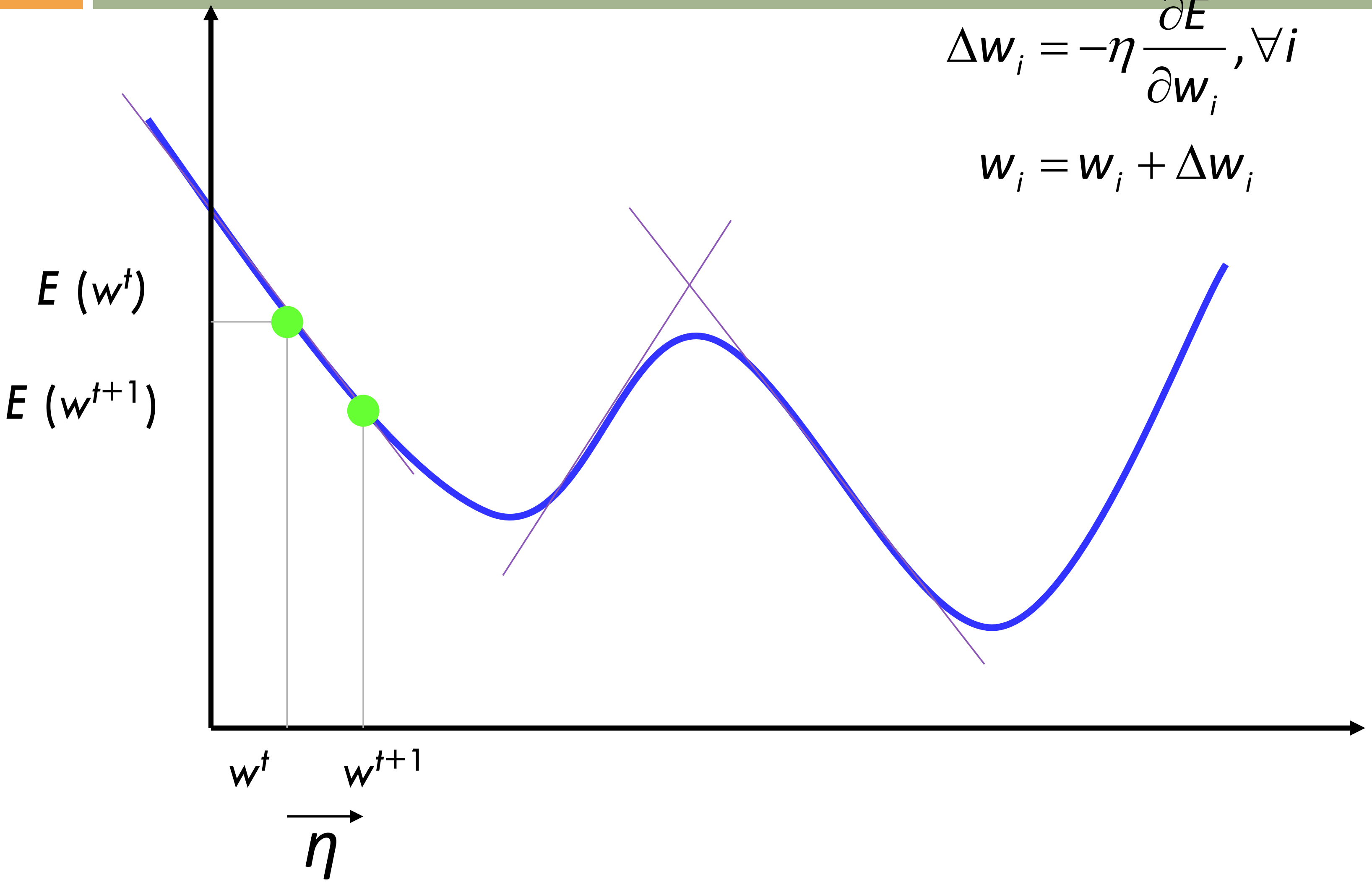
- Gradient-descent:

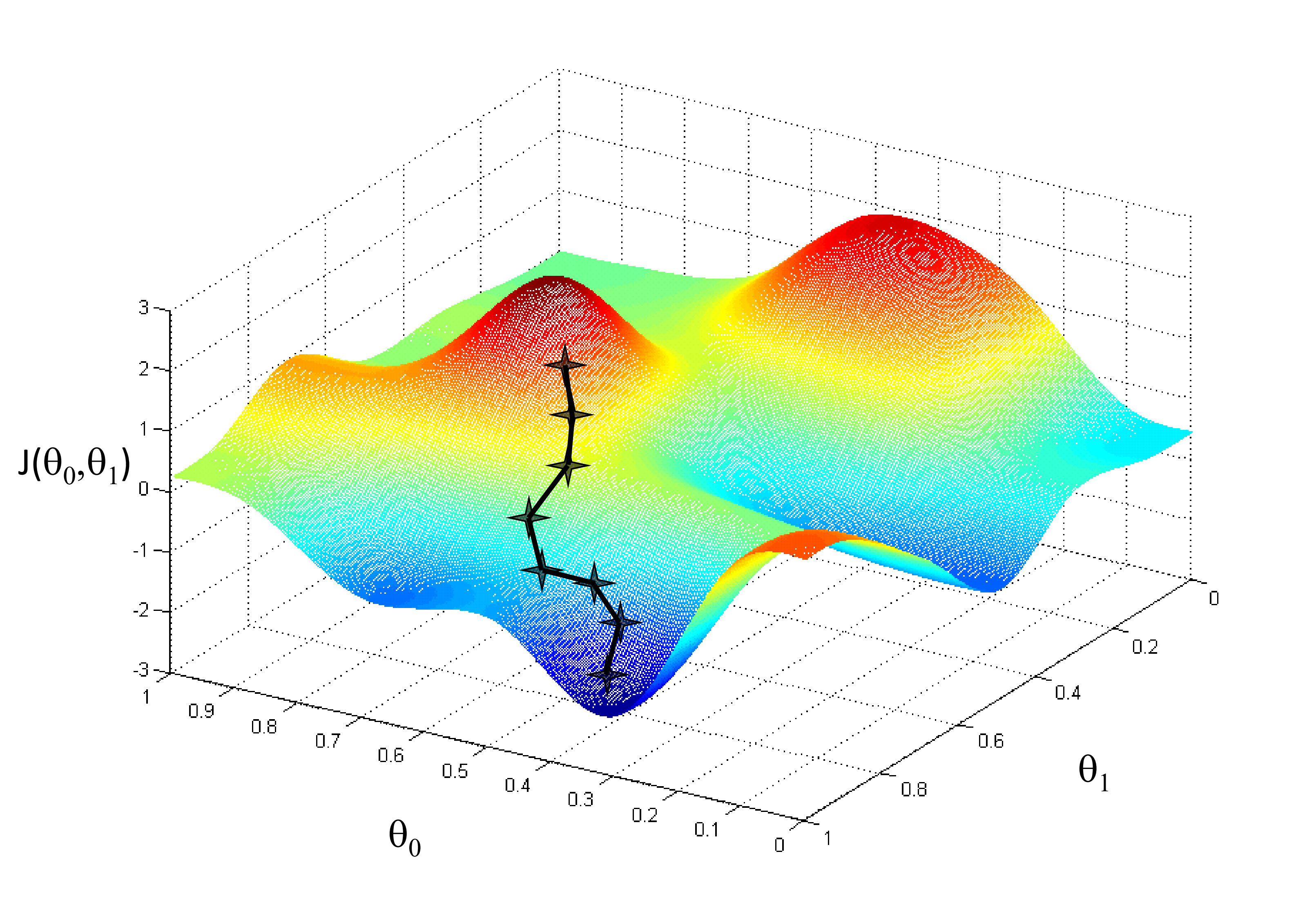
  Starts from random $\boldsymbol{w}$ and updates $\boldsymbol{w}$ iteratively in the negative direction of gradient

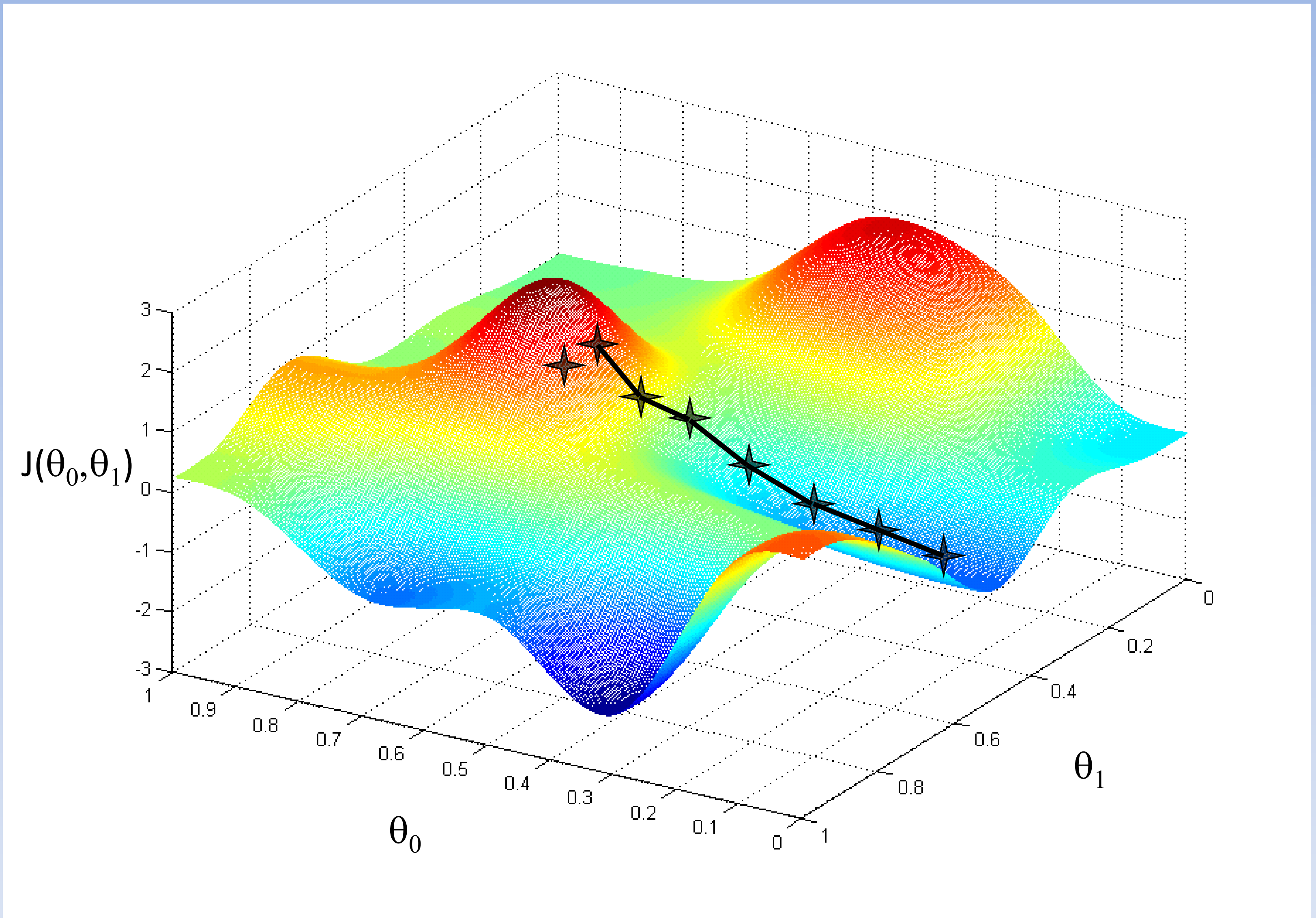
Gradient-Descent

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i$$

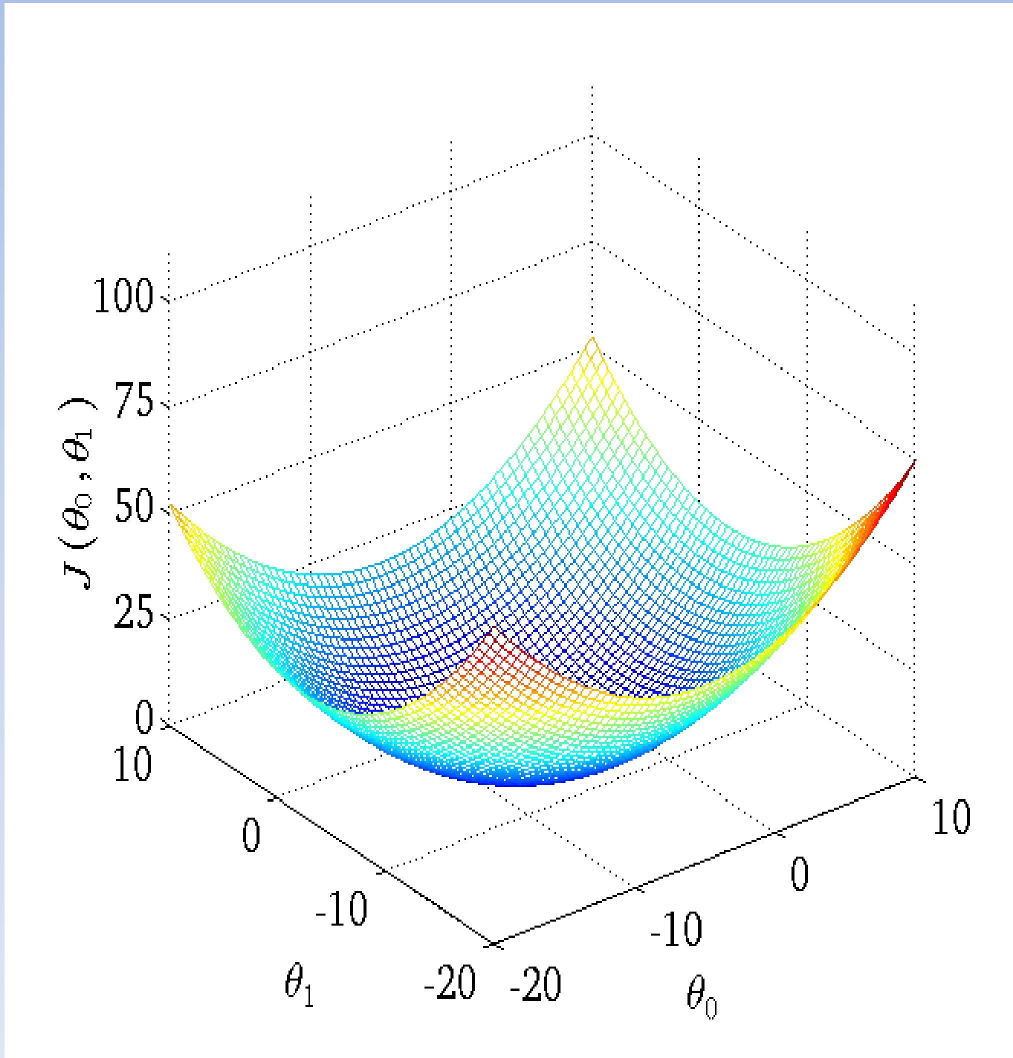
$$w_i = w_i + \Delta w_i$$




J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1

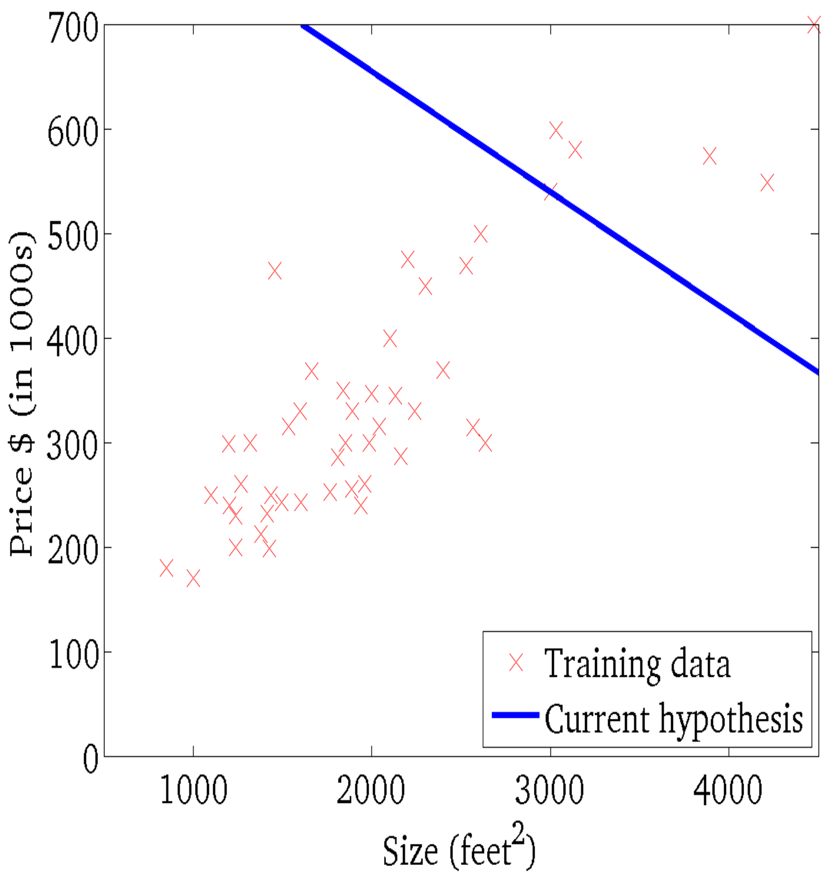

J(θ0,θ1)
3
2
1
0
-1
-2
-3
θ0
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ1
0
0.2
0.4
0.6
0.8
1

Convex Function



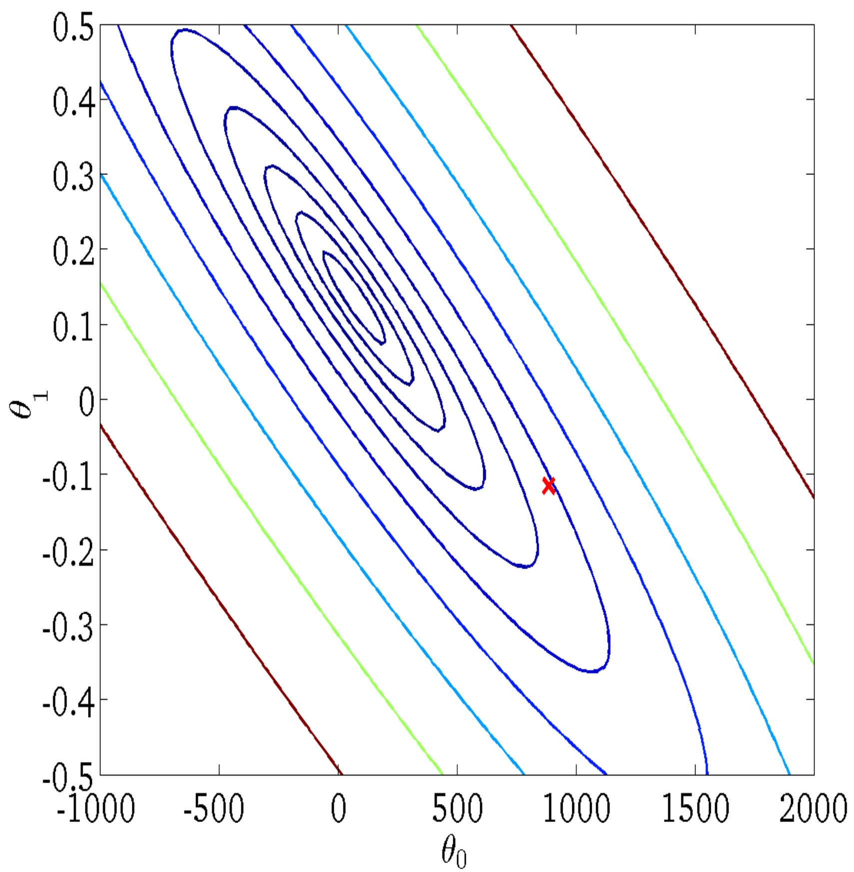
$h_\theta(x)$

(for fixed θ_0, θ_1 this is a function of x)



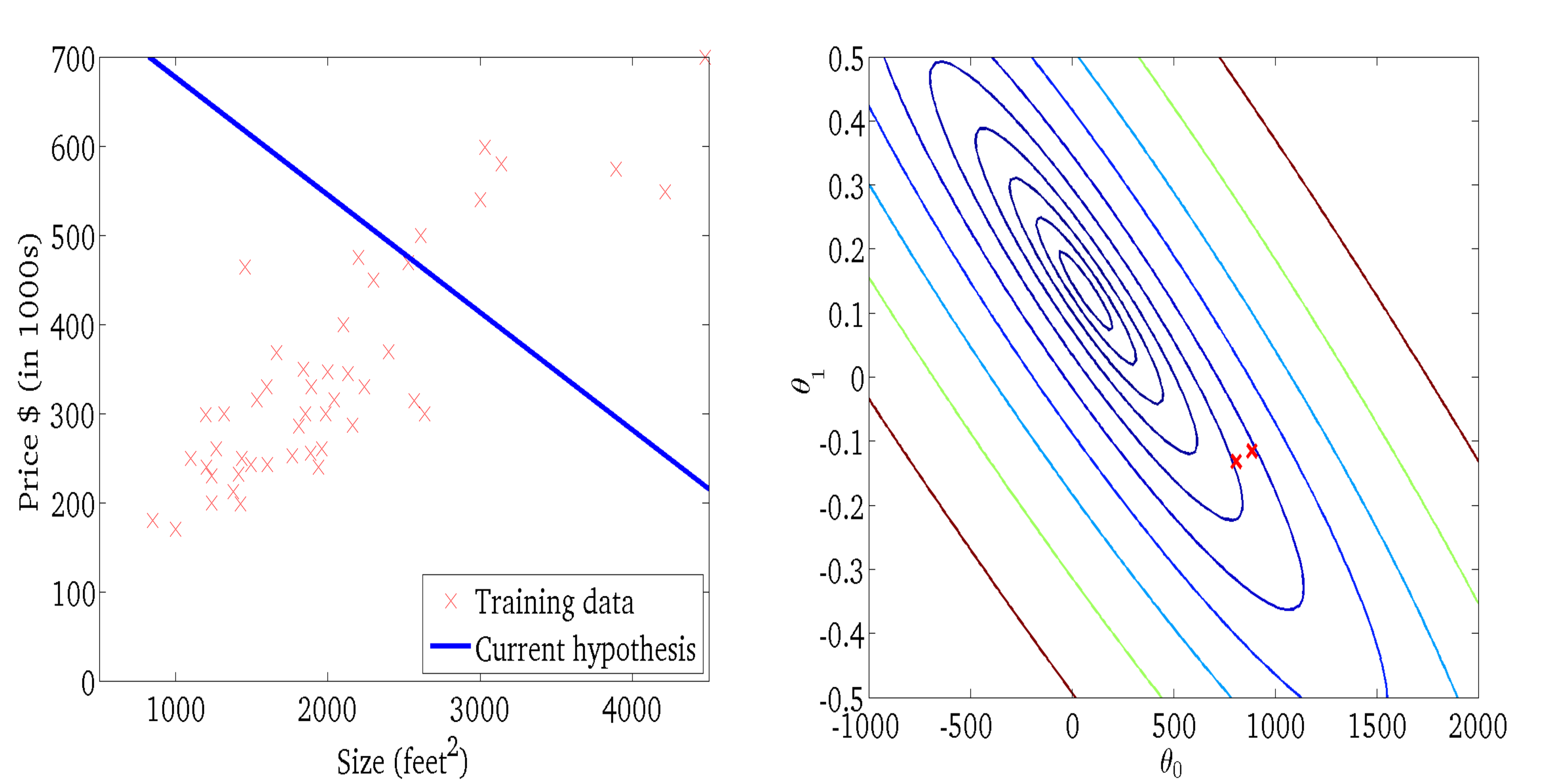
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



$h_\theta(x)$ $J(\theta_0, \theta_1)$

(for fixed θ_0, θ_1 this is a function of x) (function of the parameters θ_0, θ_1)

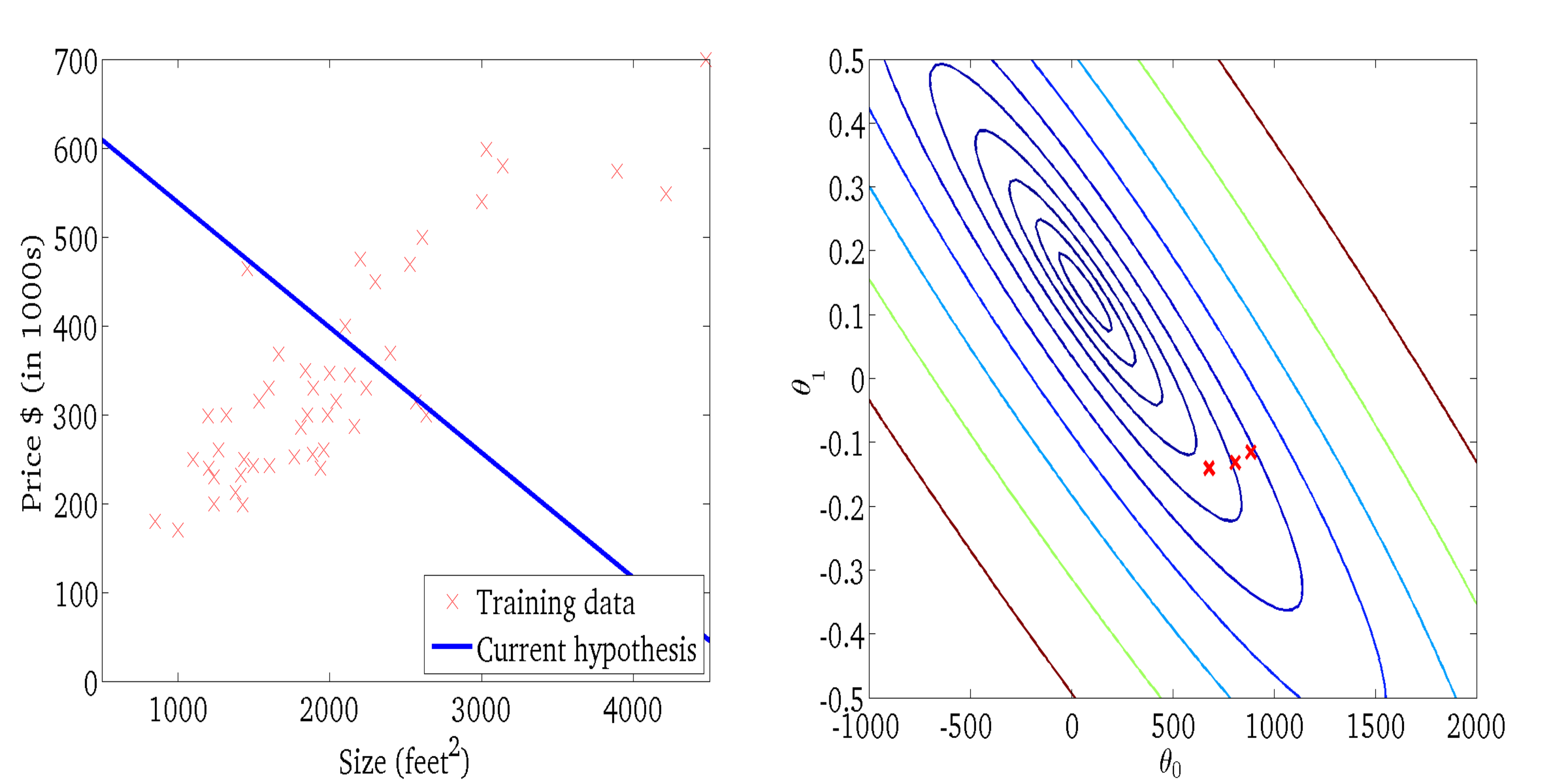


$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

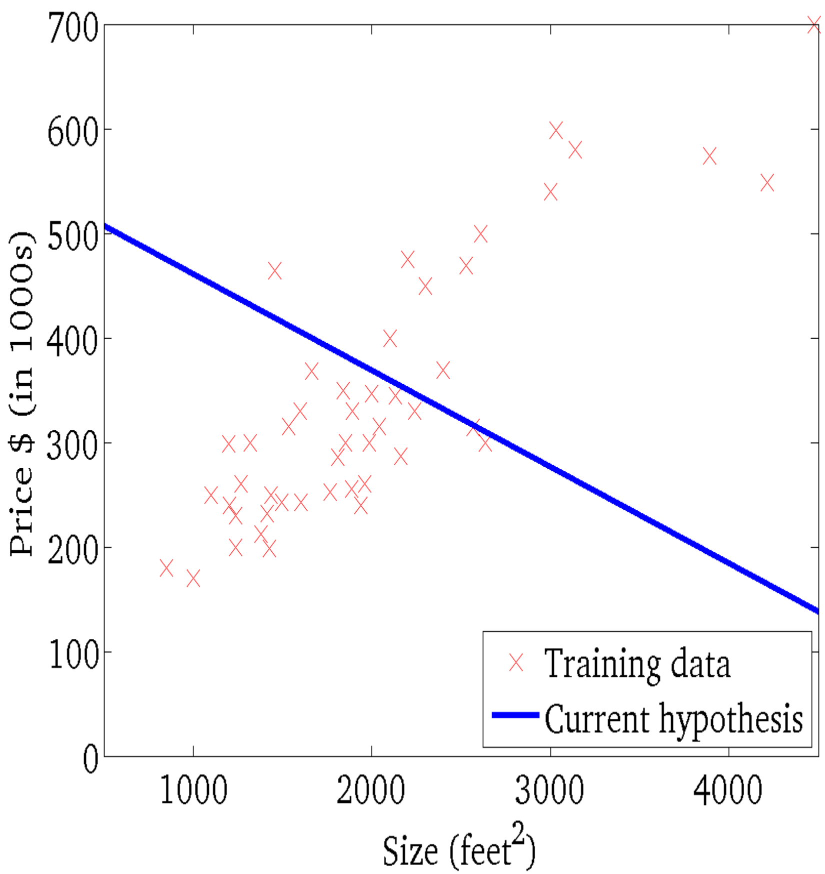


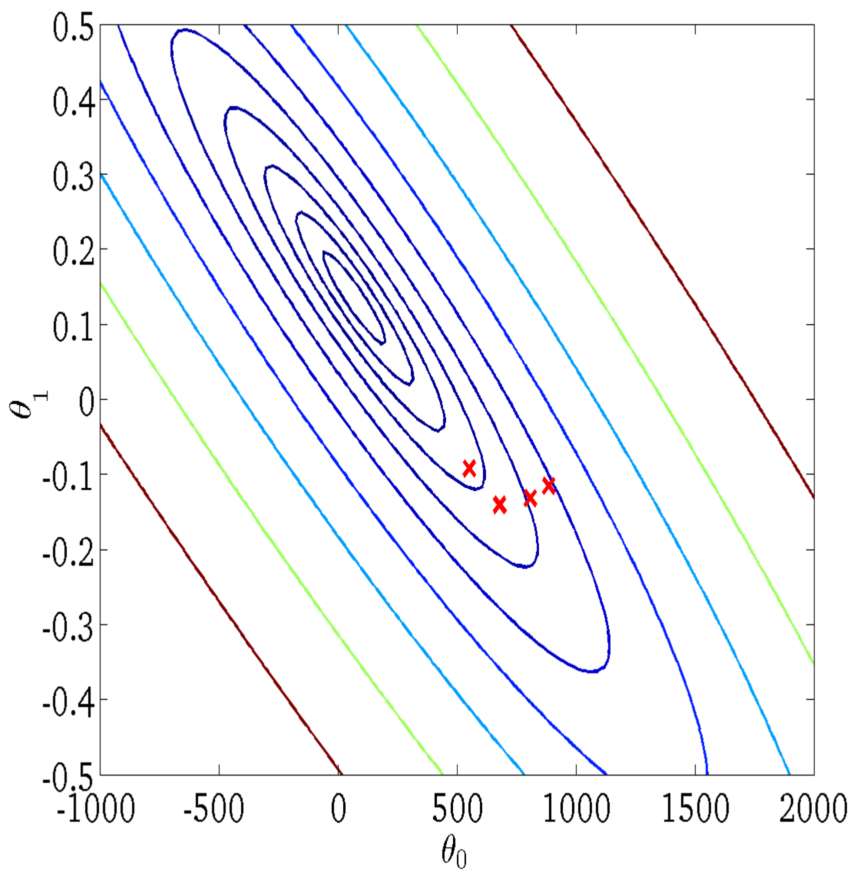
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



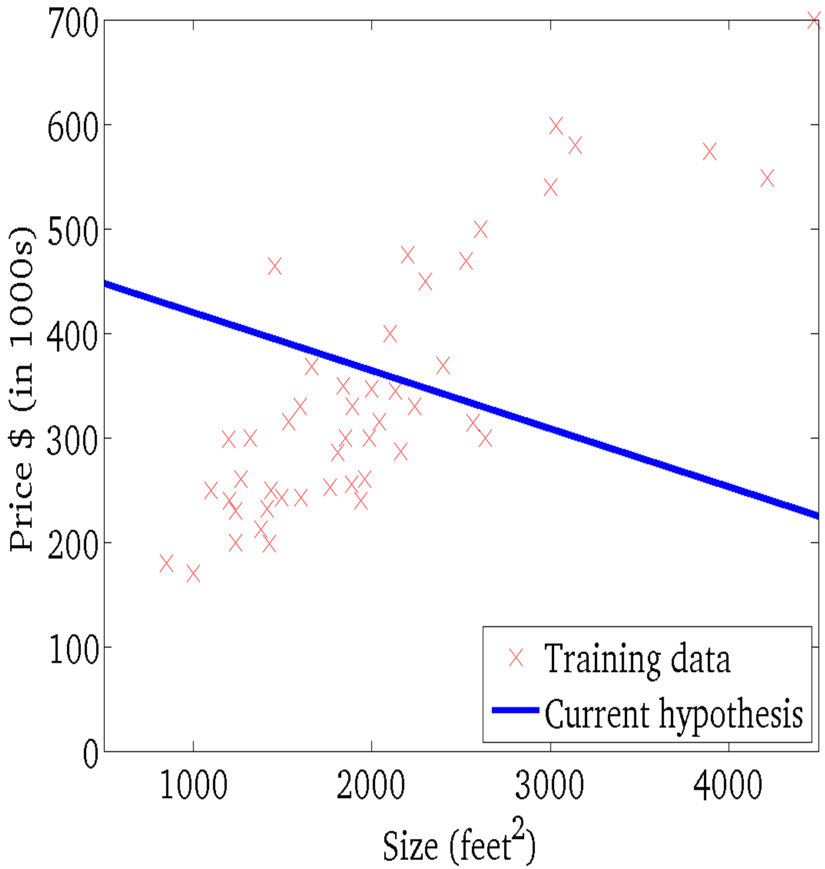


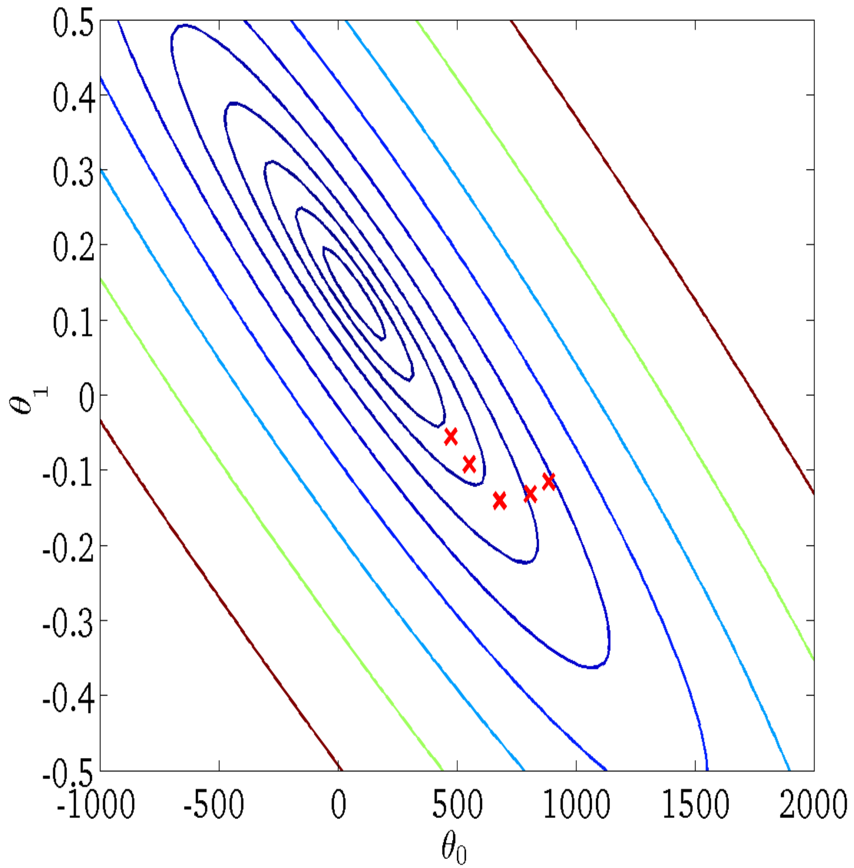
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



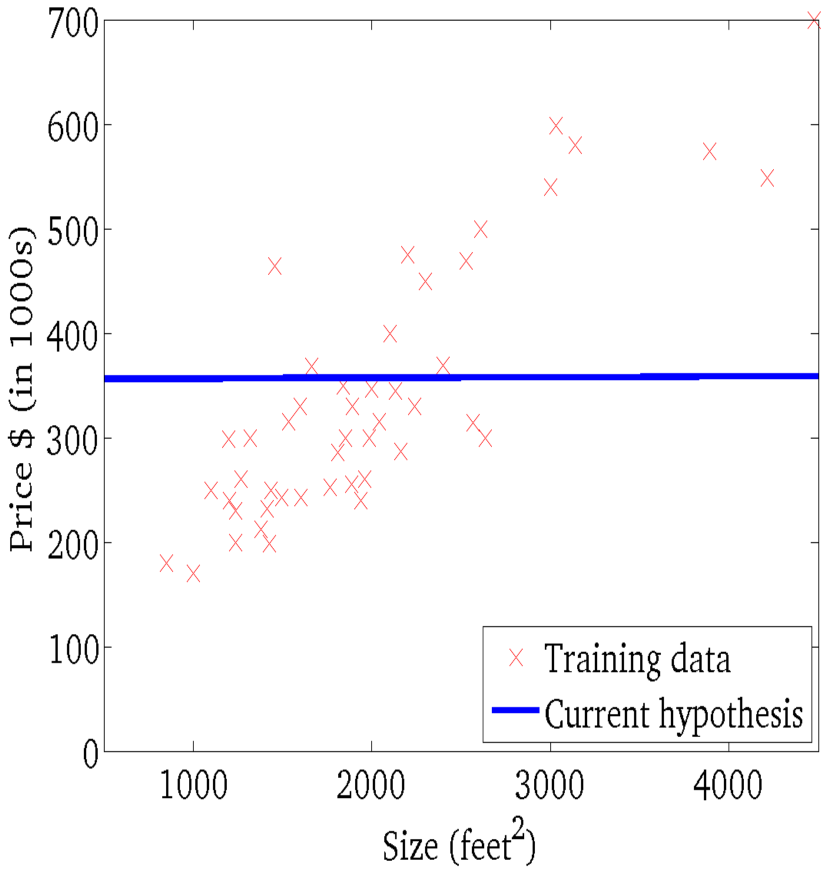


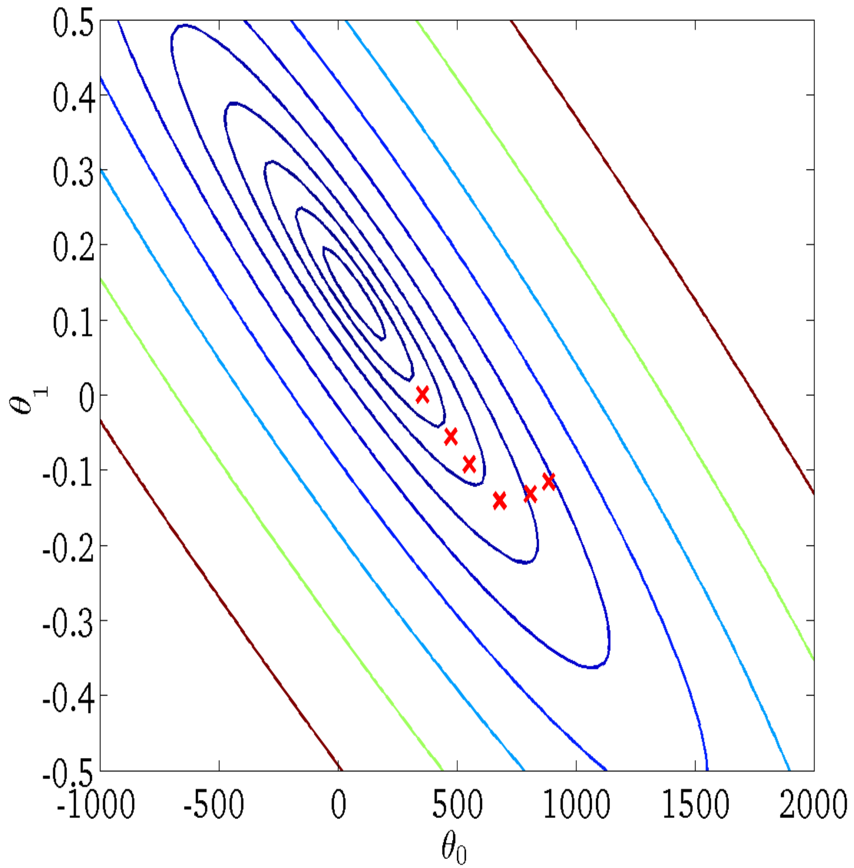
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

$J(\theta_0, \theta_1)$

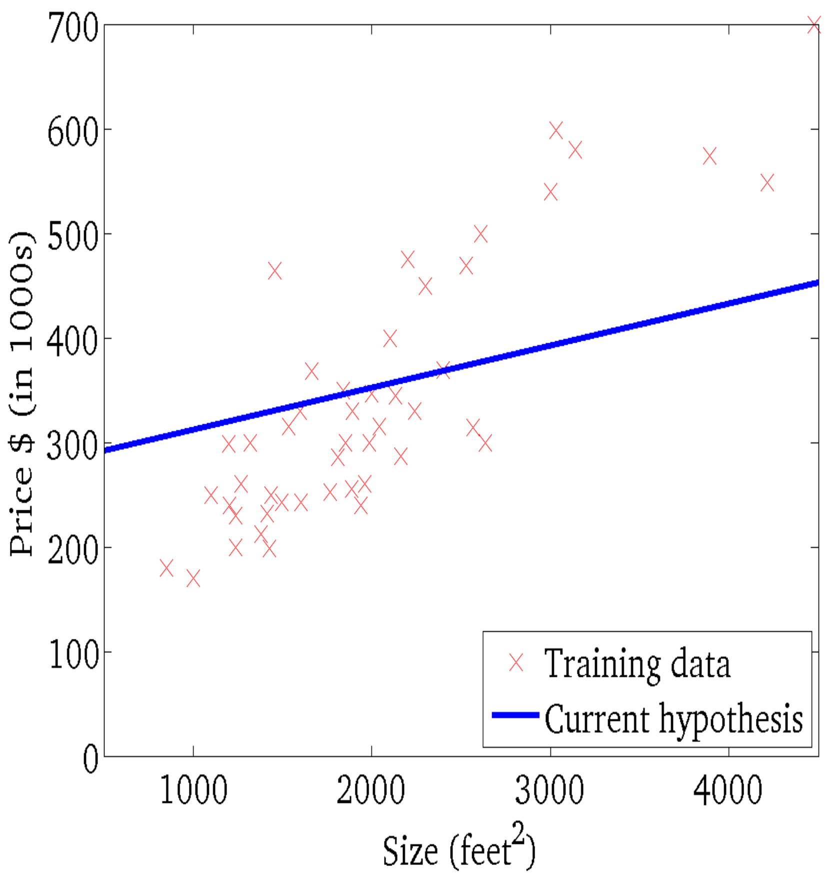
(function of the parameters θ_0, θ_1)





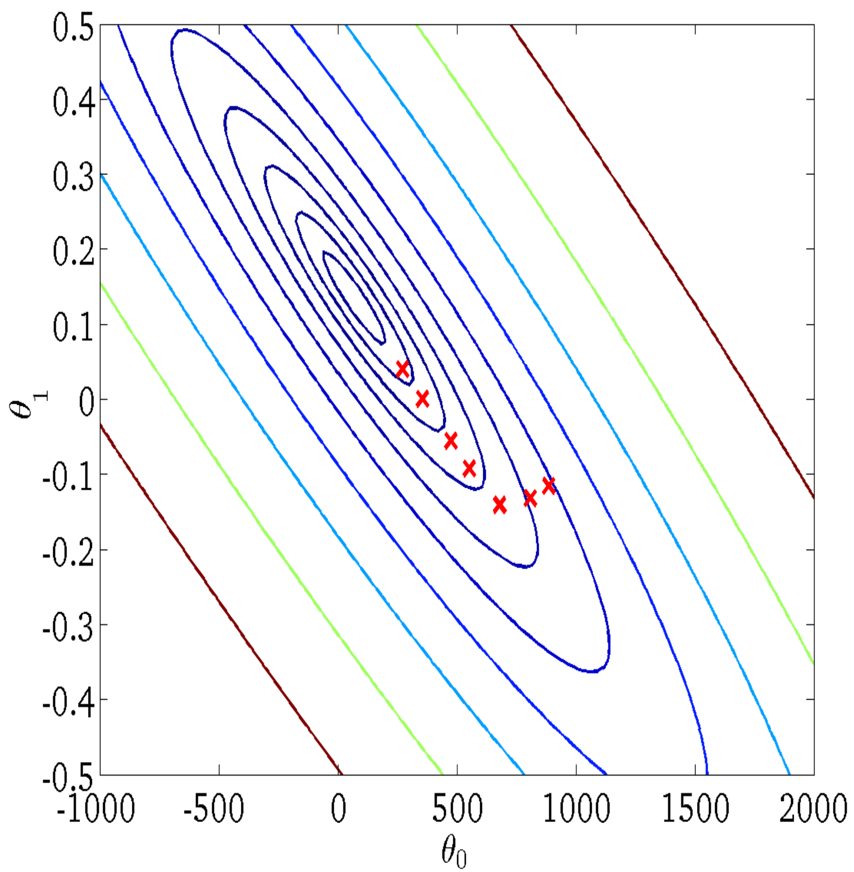
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

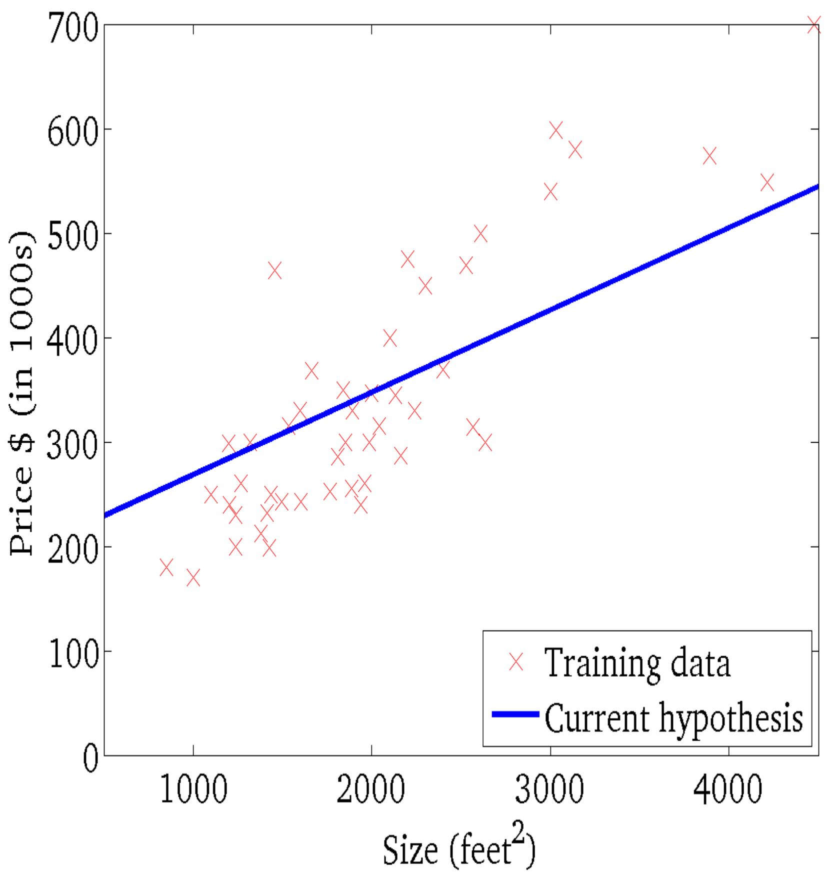


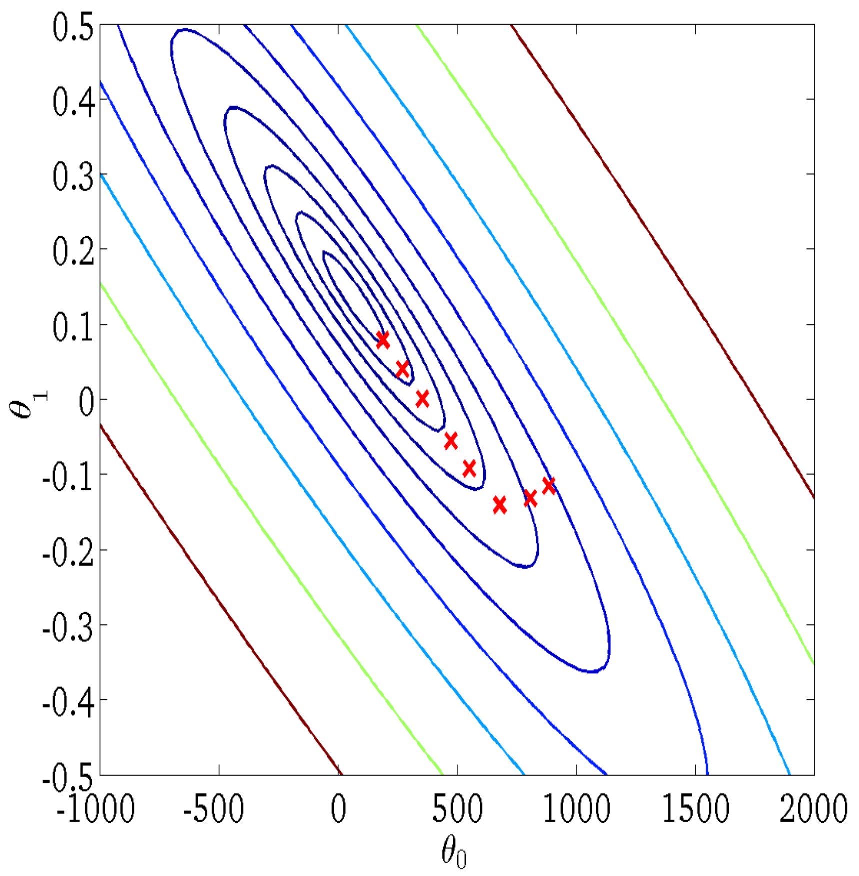
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



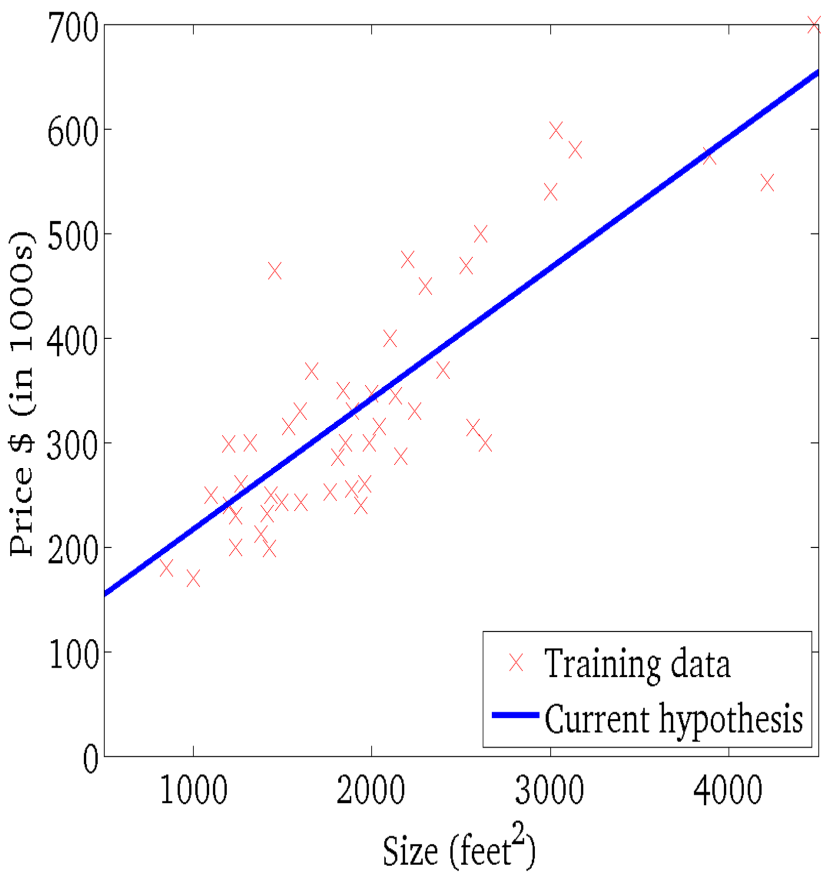


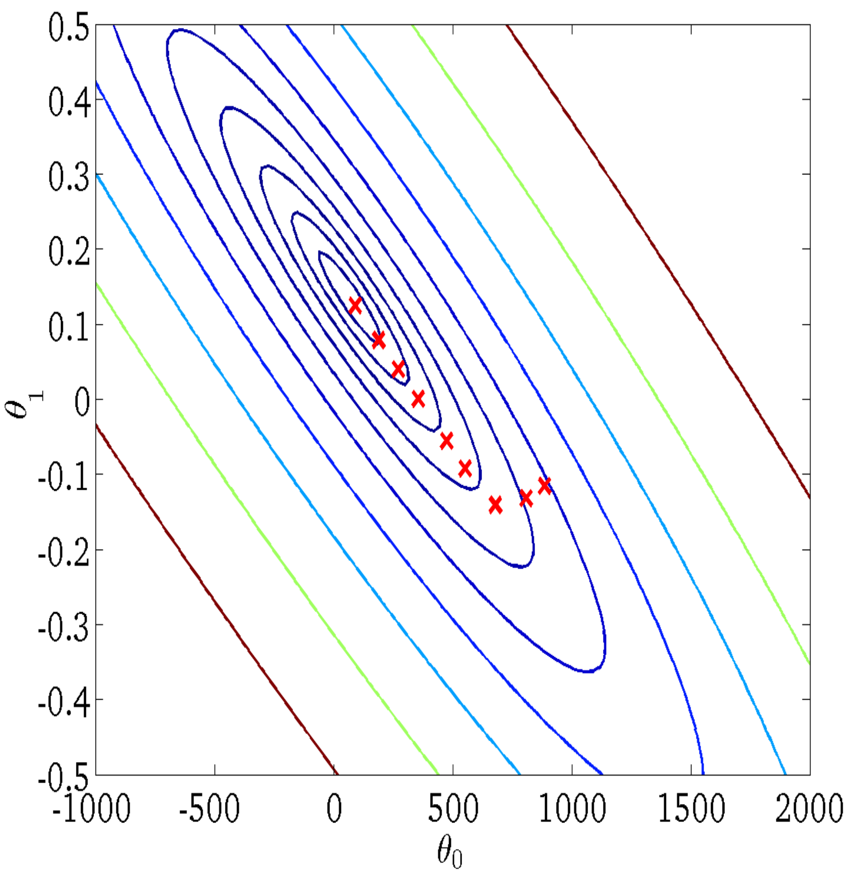
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

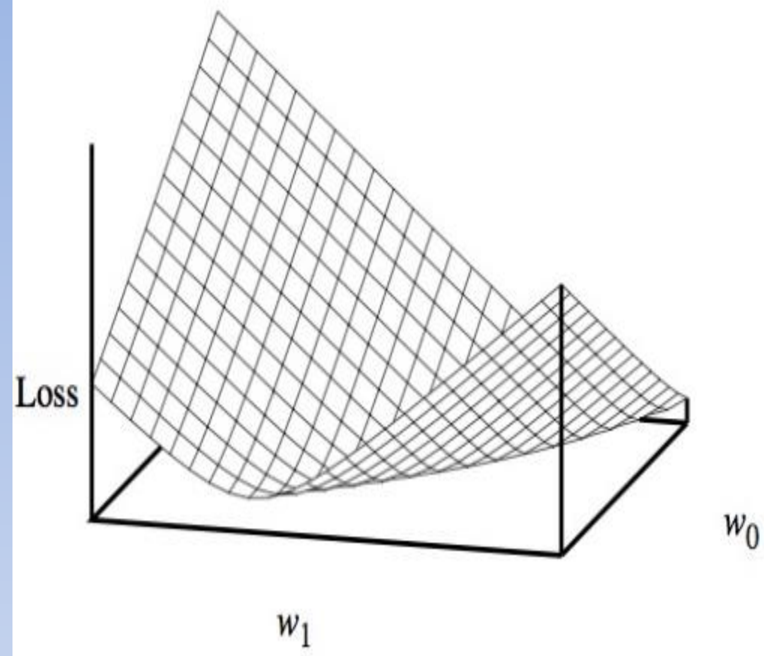
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)





Gradient Descent



(b)

w ← any point in the parameter space

loop until convergence **do**

  for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w}) \qquad (18.4)$$

Update Rule w/ Calculus

$$
\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)) \,, \qquad (18.5)
\end{aligned}
$$

$$
\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \,; \qquad \frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x
$$

Batch or Stochastic

- Batch update:
  - Use all examples to calculate delta update
  - May not be possible/practical with huge number of data points
- Stochastic update:
  - Update weights with each example
  - May not converge
- Combination:
  - Use some sample size of data points to calculate delta update

“Batch” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples.

Is this really leading to Machine Learning???????

- Are we there yet?

More than 1 Input 1 Output

- Simple extension to Gradient Descent
  - Expand to higher dimensions

18.6.2: Multivariate Linear Regression

- $h_{sw}(x\quad) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_i x_j$
  - Each example x is now a n-element vector

Multiple features (variables).

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

18.6.2: Multivariate Linear Regression

- $h_{sw(x_j)} = w_0 + w_1 x_{j,1} + w_2 x_{j,2} + \cdots + w_i x_{j,i}$
  - Each example x is now a n-element vector
- $h_{sw(x_j)} = \mathrm{w} \cdot x_j = w^T x_j = \sum_i \ w_i x_{j,i}$
- Best weights are:
  - $w^* = \underset{w}{\operatorname{argmin}} \sum_j \ L_2(y_j, \mathrm{w} \cdot x_j)$
  - Like before, Minimize square difference from actual output

Update Rule w/ Calculus

$$
\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)) , \qquad (18.5)
\end{aligned}
$$

$$
\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) ; \qquad \frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x
$$

- Generalizes to Higher Dimensions

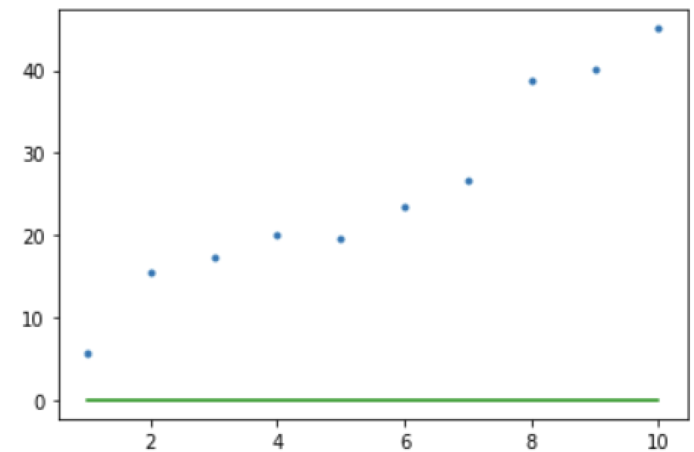
$$
w_i \leftarrow w_i + \alpha \sum_j x_{j,i}(y_j - h_{\mathbf{w}}(\mathbf{x}_j)) . \qquad (18.6)
$$

Learning Rate = 0.1

- **(x, y), H(x), y-H(x)**
- (1, 5.75) 0 5.75
- (2, 15.51) 0 15.51
- (3, 17.32) 0 17.32
- (4, 19.99) 0 19.99
- (5, 19.56) 0 19.56
- (6, 23.56) 0 23.56
- (7, 26.58) 0 26.58
- (8, 38.66) 0 38.66
- (9, 40.01) 0 40.01
- (10, 45.08) 0 45.08
- Average (y-H(x)) = 25.2
- Average (y-H(x)) * 0.1 = 2.52

```python
learningRate = 0.1
Hx = lambda w, x: w[0] + w[1]*x

w = 0, 0
count = 0
Hxs = [Hx(w,x) for x in X]
deltas = [y-h for y,h in zip(Y, Hxs)]
Error = sum([d**2 for d in deltas])
plt.plot(X, Y, '.')
plt.plot(X, Hxs, '-', color="green")
print ("Count = ", count)
print ("W = ", w)
print ("Initial Error= ", Error)

```

```
Count =  0
W =  (0, 0)
Initial Error=  7744.9708
```

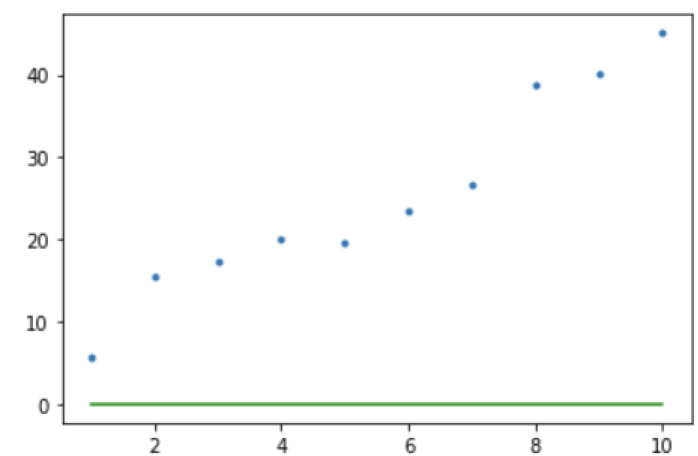


Learning Rate = 0.1

- **(x, y), H(x), x*(y-H(x))**
- (1, 5.75) 0 5.75
- (2, 15.51) 0 31.02
- (3, 17.32) 0 51.96
- (4, 19.99) 0 79.96
- (5, 19.56) 0 97.8
- (6, 23.56) 0 141.36
- (7, 26.58) 0 186.06
- (8, 38.66) 0 309.28
- (9, 40.01) 0 360.09
- (10, 45.08) 0 450.80
- Average (y-H(x))*x = 171.408
- Average (y-H(x))*x*0.1 = 17.1408

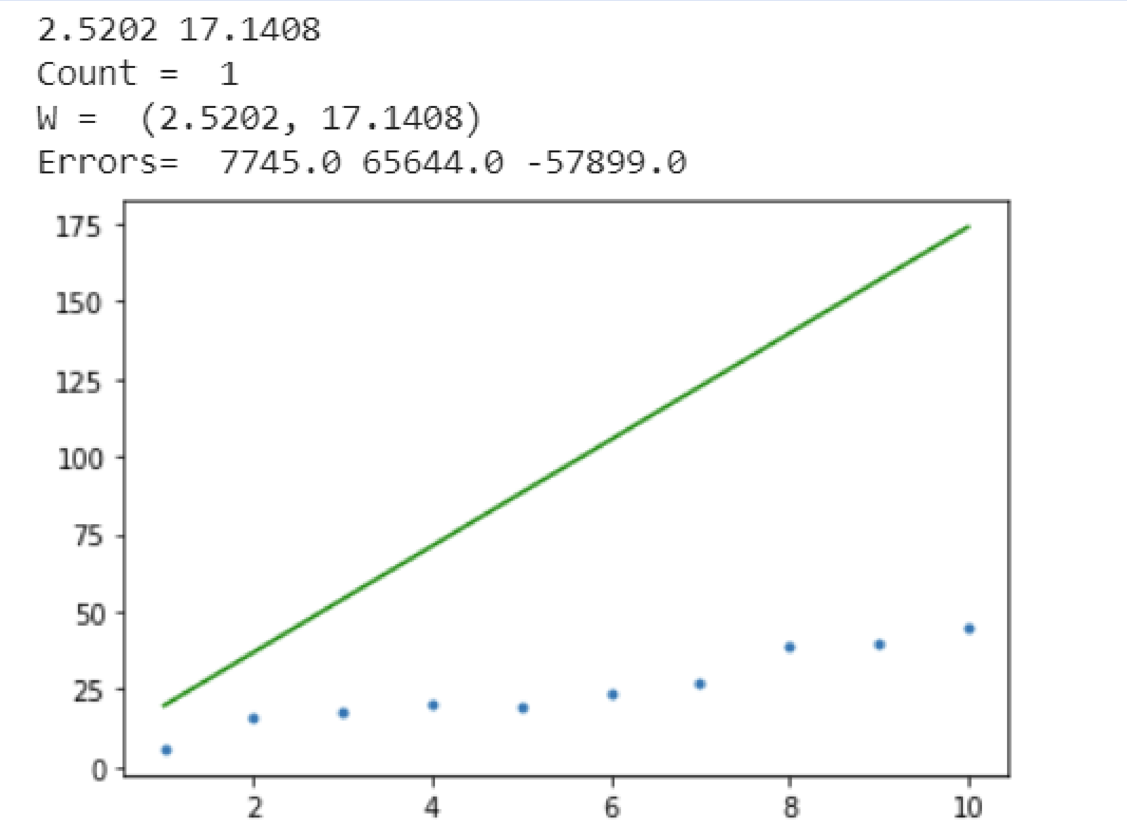
```
learningRate = 0.1
Hx = lambda w, x: w[0] + w[1]*x

w = 0, 0
count = 0
Hxs = [Hx(w,x) for x in X]
deltas = [y-h for y,h in zip(Y, Hxs)]
Error = sum([d**2 for d in deltas])
plt.plot(X, Y, '.')
plt.plot(X, Hxs, '-', color="green")
print ("Count = ", count)
print ("W = ", w)
print ("Initial Error= ", Error)

```

```
Count =  0
W =  (0, 0)
Initial Error=  7744.9708
```



Learning Rate = 0.1
1 Update

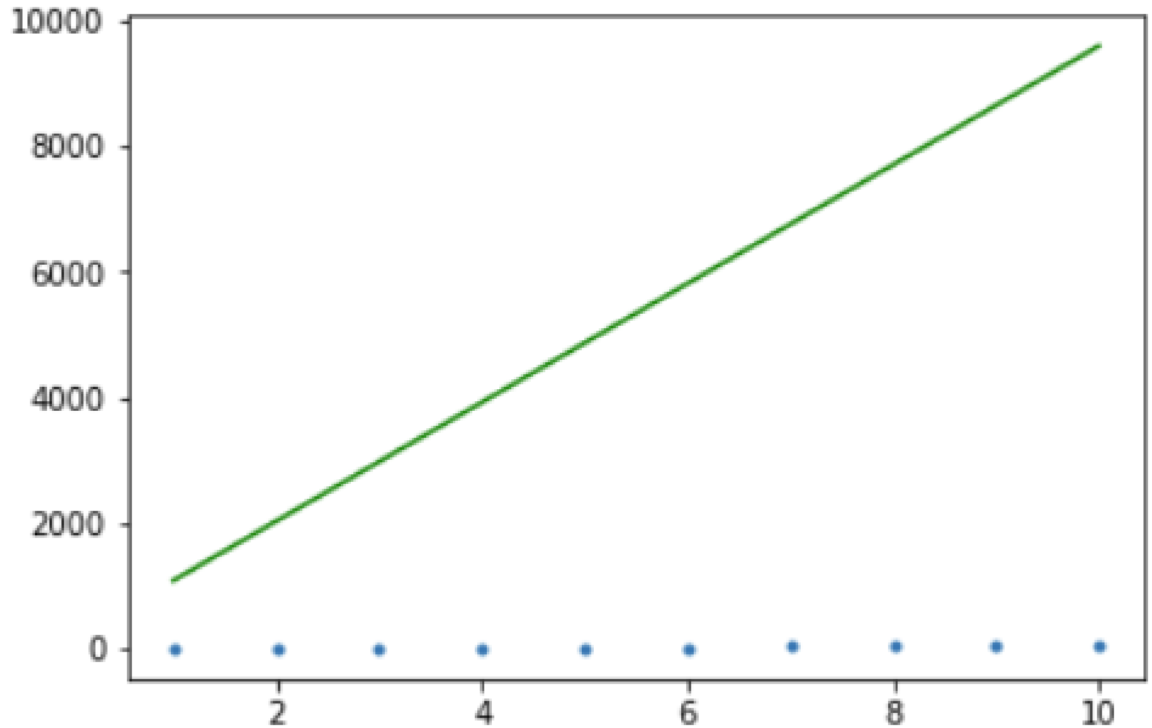
- Error has gone from 7745 to 65644



5 Updates – Only Worse

- Error now 355026474

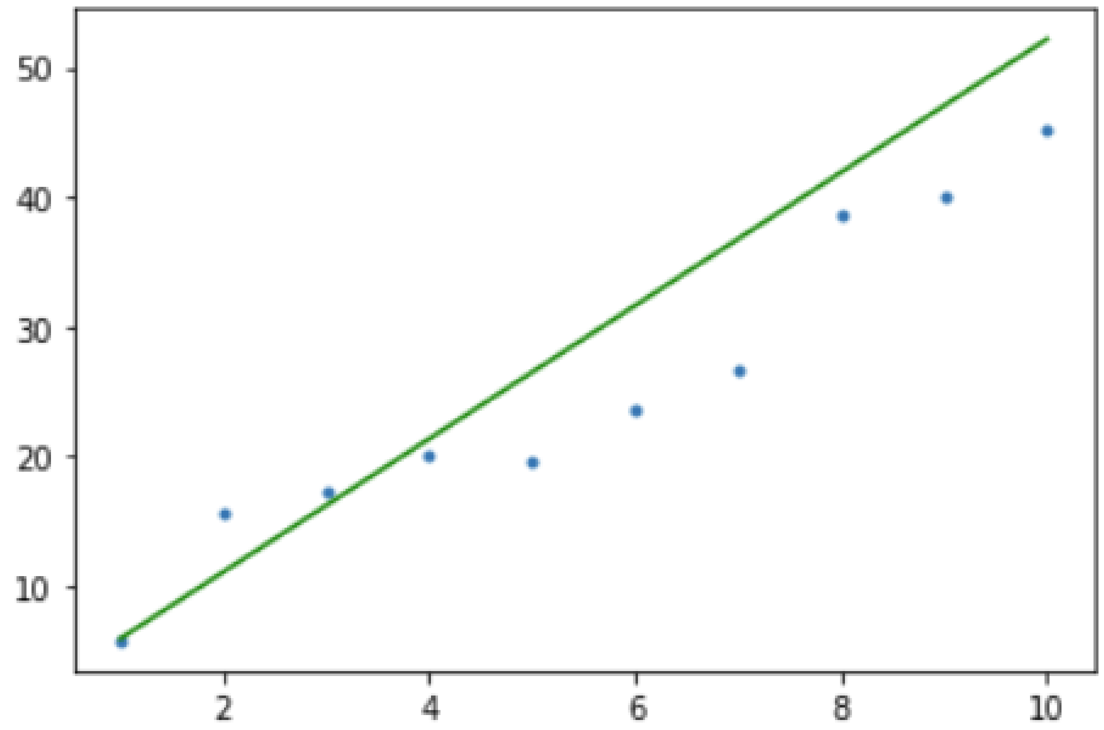
```
181.35072463249998 1262.1600563737495
Count =  5
W =  (136.05461760749998, 945.2493037737496)
Errors=  41383028.0 355026474.0 -313643446.0
```



Learning Rate = 0.03
1 Update

- Error 7745 to 349

```
0.75606 5.142239999999999
Count =  1
W =  (0.75606, 5.142239999999999)
Errors=  7745.0 349.0 7396.0
```

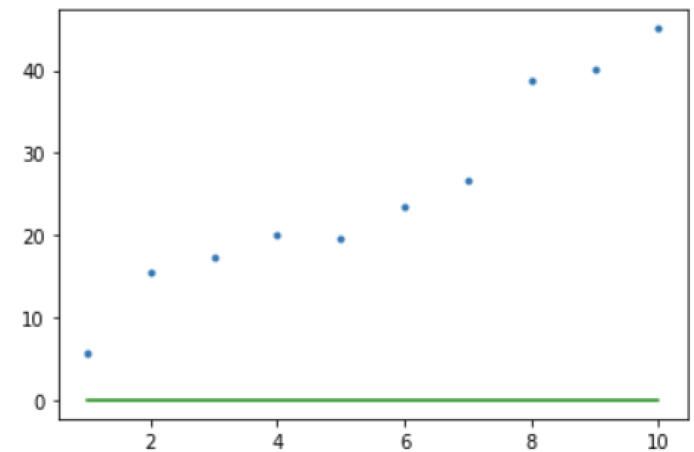


Learning Rate = 0.03

- **(x, y), H(x), y-H(x)**
- (1, 5.75) 0 5.75
- (2, 15.51) 0 15.51
- (3, 17.32) 0 17.32
- (4, 19.99) 0 19.99
- (5, 19.56) 0 19.56
- (6, 23.56) 0 23.56
- (7, 26.58) 0 26.58
- (8, 38.66) 0 38.66
- (9, 40.01) 0 40.01
- (10, 45.08) 0 45.08
- Average (y-H(x)) = 25.2
- Average (y-H(x)) * 0.03 = 0.756

```python
learningRate = 0.1
Hx = lambda w, x: w[0] + w[1]*x

w = 0, 0
count = 0
Hxs = [Hx(w,x) for x in X]
deltas = [y-h for y,h in zip(Y, Hxs)]
Error = sum([d**2 for d in deltas])
plt.plot(X, Y, '.')
plt.plot(X, Hxs, '-', color="green")
print ("Count = ", count)
print ("W = ", w)
print ("Initial Error= ", Error)

```

```
Count =  0
W =  (0, 0)
Initial Error=  7744.9708
```

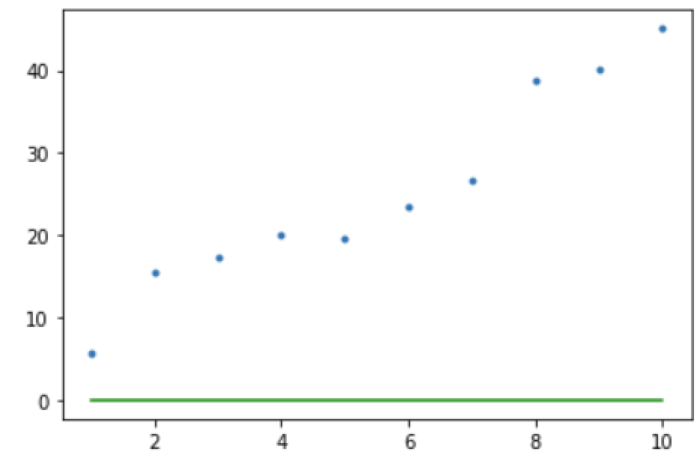


Learning Rate = 0.03

- **(x, y), H(x), x*(y-H(x))**
- (1, 5.75) 0 5.75
- (2, 15.51) 0 31.02
- (3, 17.32) 0 51.96
- (4, 19.99) 0 79.96
- (5, 19.56) 0 97.8
- (6, 23.56) 0 141.36
- (7, 26.58) 0 186.06
- (8, 38.66) 0 309.28
- (9, 40.01) 0 360.09
- (10, 45.08) 0 450.80
- Average (y-H(x))*x = 171.408
- Average (y-H(x))*x*0.03 = 5.14

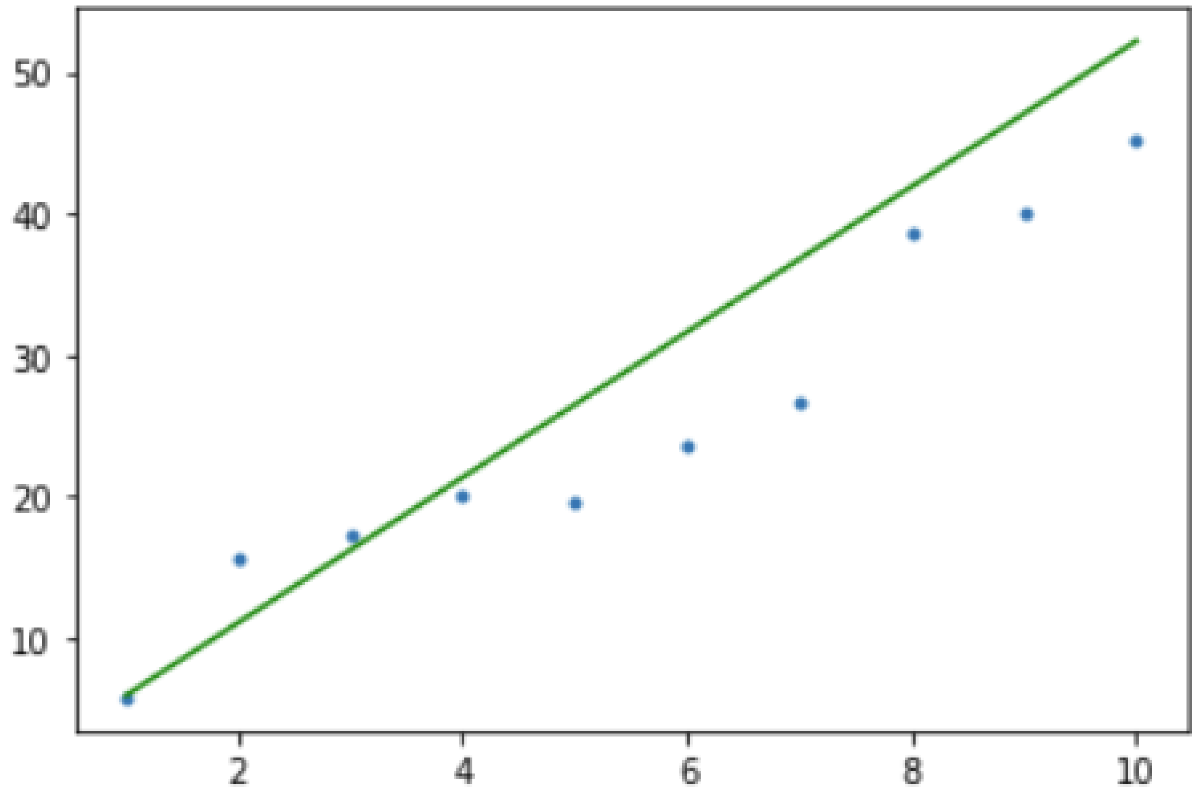
```
learningRate = 0.1
Hx = lambda w, x: w[0] + w[1]*x

w = 0, 0
count = 0
Hxs = [Hx(w,x) for x in X]
deltas = [y-h for y,h in zip(Y, Hxs)]
Error = sum([d**2 for d in deltas])
plt.plot(X, Y, '.')
plt.plot(X, Hxs, '-', color="green")
print ("Count = ", count)
print ("W = ", w)
print ("Initial Error= ", Error)

```

```
Count =  0
W =  (0, 0)
Initial Error=  7744.9708
```



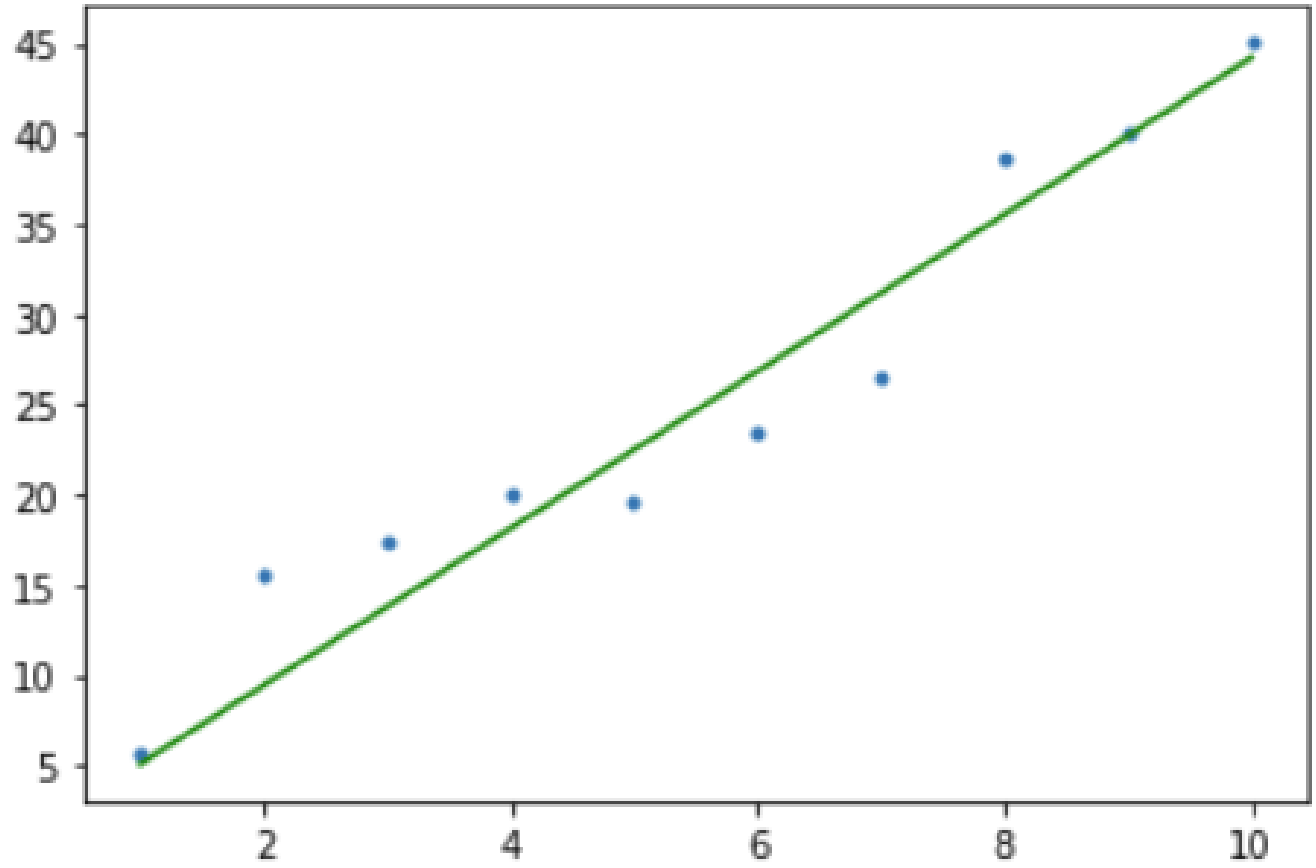
After 1 Iteration

```
0.75606 5.142239999999999
Count =  1
W =  (0.75606, 5.142239999999999)
Errors=  7744.9708 349.2571 7395.7137
```



Now After 5 Iterations

```
0.01740098487997498З 0.002855196170212З485
Count =  5
W =  (0.711363251777475, 4.353401542310213)
Errors=  104.7282 104.5325 0.1957
```



Learn Function:

$F(x) = 3 + 4x$

x	F(x)
1	7
2	11
3	15
4	19
5	23

Learn Function:
F(x) = 3 + 4x

x	F(x)	$h_w(x)$
1	7	0
2	11	0
3	15	0
4	19	0
5	23	0

$$h_w(x) = w_0 + w_1 x$$

$$w_0 = 0$$

$$w_1 = 0$$

How good is this hypothesis?

Learn Function:
F(x) = 3 + 4x

x	F(x)	$h_w(x)$
1	7	4
2	11	6
3	15	8
4	19	10
5	23	12

$$h_w(x) = w_0 + w_1 x$$
$$w_0 = 2$$
$$w_1 = 2$$

How good is this hypothesis?

Learn Function:
F(x) = 3 + 4x

x	F(x)	$h_w(x)$	$\boldsymbol{Error}$
1	7	4	$(7-4)^2 = 9$
2	11	6	$(11-6)^2 = 25$
3	15	8	$(15-8)^2 = 49$
4	19	10	$(19-10)^2 = 81$
5	23	12	$(23-12)^2 = 121$
$\frac{1}{m}\sum_i \left(F(x_i) - h_\theta(x_i)\right)^2 =$			285

$$h_\theta(x) = w_0 + w_1 x$$
$$w_0 = 2$$
$$w_1 = 2$$

How good is this hypothesis?

Learn Function:F(x) = 3 + 4x

x	F(x)	$h_w(x)$	$Error$	$\frac{\partial}{\partial Error} = x(F(x_i) - h_w(x_i))$
1	7	4	$(7-4)^2 = 9$	$1 * (7-4) \quad = 3$
2	11	6	$(11-6)^2 = 25$	$2 * (11-6) \quad = 10$
3	15	8	$(15-8)^2 = 49$	$3 * (15-8) \quad = 21$
4	19	10	$(19-10)^2 = 81$	$4 * (19-10) \quad = 36$
5	23	12	$(23-12)^2 = 121$	$5 * (23-12) \quad = 55$
$\frac{1}{m}\sum_i \left(F(x_i) - h_w(x_i)\right)^2 =$				$\frac{1}{m}\sum_i x_i\left(F(x_i) - h_w(x_i)\right) \quad =$

$$h_\theta(x) = \ w_0 + w_1 x$$

$$w_0 = 2$$

$$w_1 = 2$$

Adjust Hypothesis

Learn Function: F(x) = 3 + 4x

x	F(x)	$h_w(x)$	$Error$	$\frac{\partial}{\partial Error} = x(F(x_i) - h_\theta(x_i))$
1	7	6	$(7-4)^2 = 9$	$1 * (7-4) \quad = 3$
2	10	8	$(11-6)^2 = 25$	$2 * (11-6) \quad = 10$
3	13	10	$(15-8)^2 = 49$	$3 * (15-8) \quad = 21$
4	16	12	$(19-10)^2 = 81$	$4 * (19-10) \quad = 36$
5	19	14	$(23-12)^2 = 121$	$5 * (23-12) \quad = 55$
$\frac{1}{m}\sum_i \left(F(x_i) - h_w(x_i)\right)^2 =$				$\frac{1}{m}\sum_i x_i\left(F(x_i) - h_w(x_i)\right) \quad =$

$$h_w(x) = w_0 + w_1 x$$
$$w_0 = 2$$
$$w_1 = 2$$

Adjust Hypothesis w/ Gradient Descent

x_0	x	F(x)	$h_w(x)$	$Error$	$\frac{\partial}{\partial J(\boldsymbol{w}_0)} = x(F(x_i) - h_w(x_i))$	$\frac{\partial}{\partial J(\boldsymbol{w}_1)} = x(h_w(x) - \mathbf{F}(\mathbf{x}))$
1	1	7	6	$(7-4)^2 = 9$	$1 * (7-4) \quad = 3$	$1 * (7-4) \quad = 3$
1	2	10	8	$(11-6)^2 = 25$	$1 * (11-6) \quad = 10$	$2 * (11-6) \quad = 10$
1	3	13	10	$(15-8)^2 = 49$	$1 * (15-8) \quad = 21$	$3 * (15-8) \quad = 21$
1	4	16	12	$(19-10)^2 = 81$	$1 * (19-10) \quad = 36$	$4 * (19-10) \quad = 36$
1	5	19	14	$(23-12)^2 = 121$	$1 * (23-12) \quad = 55$	$5 * (23-12) \quad = 55$
$\frac{1}{m}\sum_i \left(F(x_i) - h_w(x_i)\right)^2$					$\frac{1}{m}\sum_i 1 * \left(F(x_i) - h_w(x_i)\right)$ = 7	$\frac{1}{m}\sum_i x_i\left(F(x_i) - h_w(x_i)\right)$ = 25

$$h_\theta(x) = w_0 + w_1 x$$
$$w_0 = 2$$
$$w_1 = 2$$

Adjust Hypothesis w/ Gradient Descent

Learning Rate = 1

- $w_0 = 2 + 7 = 9$
- $w_1 = 2 + 25 = 27$
- New Error=33415.0

Learning Rate = 0.1

- $w_0 = 2 + 0.7 = 2.7$
- $w_1 = 2 + 2.5 =$4.5
- New Error=9.7

Learning Rate = 0.03

- $w_0 = 2 + 0.21 = 2.21$
- $w_1 = 2 + 0.75 = 2.75$
- New Error=118.68

Gradient Descent w/ Feature Scaling

- With Multiple features, scale matters!
- Performance improvement by adjusting features to equal scale.

Mean Normalization

x_1	**Mean Normalized** x_1
89	(89-81) = 8
72	(72-81) = -9
94	(94-81) = 13
69	(69-81) = -12
$Total = 324$	Total = 0

$$\mu = \frac{324}{4} = 81$$

$$Mean\ Normalized\ x_1 = x_1 - \mu$$

Feature Scaling

x_1	**Feature Scaled** x_1
89	89/25=3.56
72	72/25=2.88
94	94/25=3.76
69	69/25=2.76
$Max(x_1) - Min(x_1) = 94 - 69 = 25$	

$$Max(x_1) - Min(x_1) = (94 - 69) = 25$$

Feature Scaled $x_1 = \frac{x_1}{Max(x_1) - Min(x_1)}$

Feature Scaled/Mean Normalization

x_1	**Mean Normalized** x_1	**Feature Scaled/ Mean Normalized** x_1
89	(89-81) = 8	8/25=0.32
72	(72-81) = -9	-9/25=-0.36
94	(94-81) = 13	13/25=0.52
69	(69-81) = -12	-12/25=-0.48
$Total = 324$	Total = 0	

$$\mu = \frac{324}{4} = 81$$

$$Mean\ Normalized\ x_1 = x_1 - \mu$$

$$Max(x_1) - Min(x_1) = (13 - (-12)) = 25$$

Update Rule w/ Calculus

$$
\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\
&= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)) \,, \qquad (18.5)
\end{aligned}
$$

$$
\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \,; \qquad \frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x
$$

- Generalizes to Higher Dimensions

$$
w_i \leftarrow w_i + \alpha \sum_j x_{j,i}(y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \,. \qquad (18.6)
$$

Linear Regression & Overfitting!

- We can minimize squared error:
  - $w^* = (X^T X)^{-1} X^T y$
  - Now possibly overfitting in some dimensions
- Regularization: Common approach to overfitting problem
  - Penalize complexity in Cost Function!

Linear Regression & Regularization

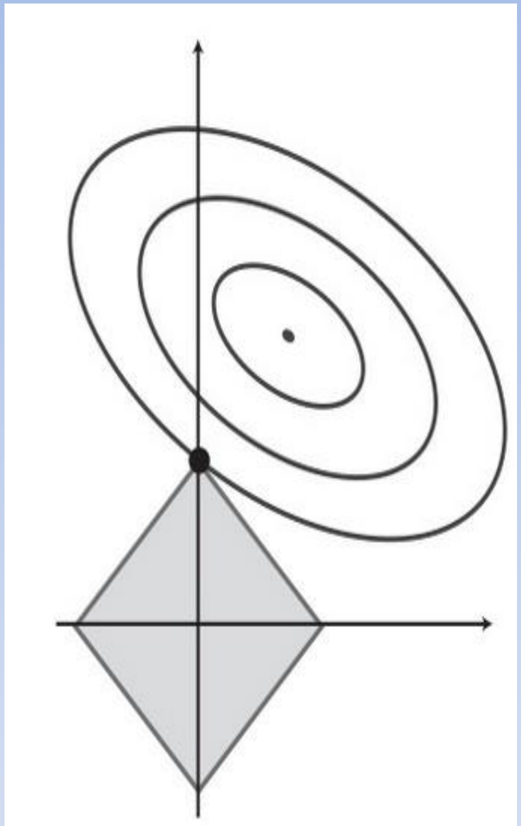
- Cost(Hypothesis) = Sum of :

  + EmpiricalLoss(Hypothesis)

  + λ*Complexity(Hypothesis)

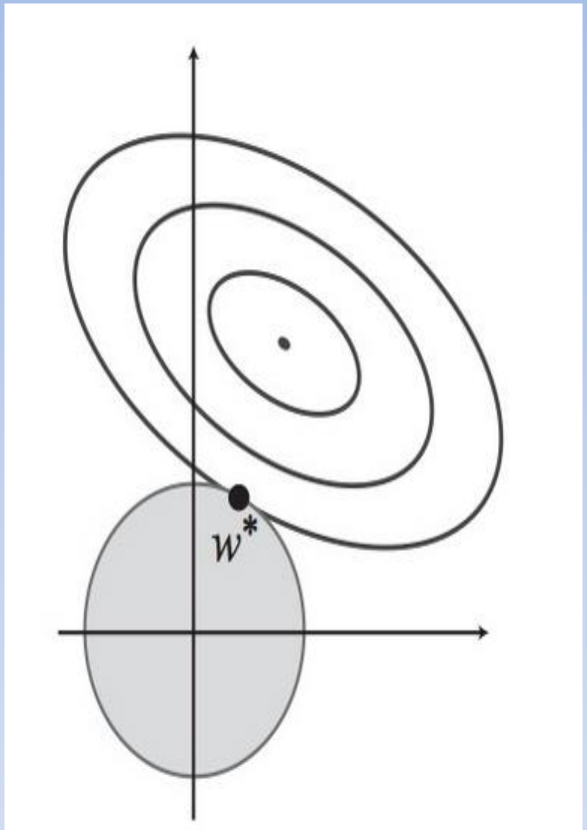
- Complexity(Hypothesis) =
  - $L_q(w) = \sum_i |w_i|^q$
  - Sum out the weights!
  - Prefer smaller weights

L_1 versus L_2



L_1 Regularization

- Produces Sparse Model

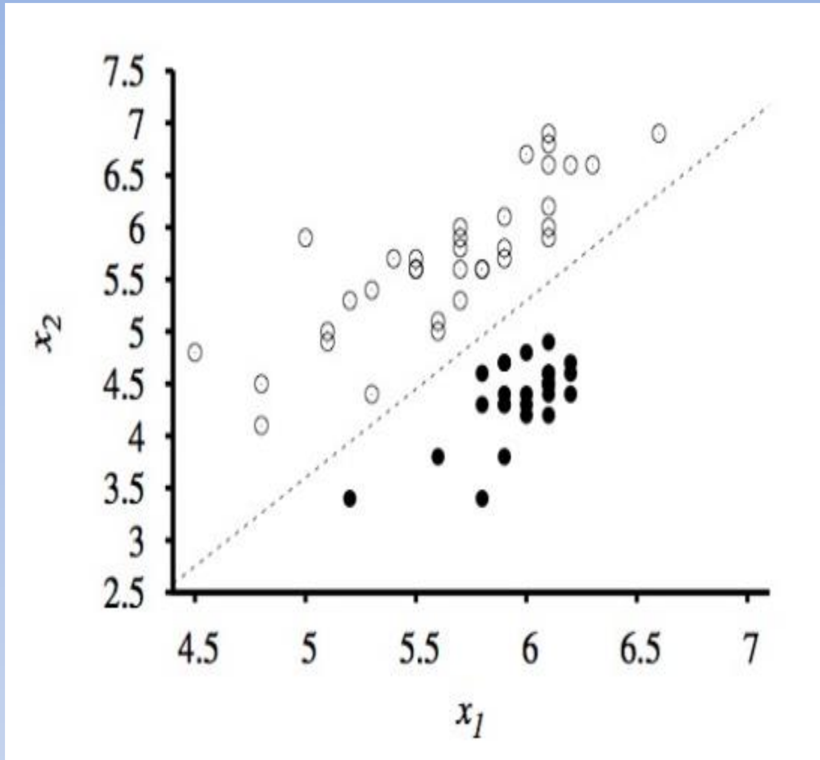


L_2 Regularization

Is this leading to Machine Learning??

- Supervised Learning – Classification
  - Set of Examples
  - Predict Class with new examples
- Can Linear Regression Help?????
- How can we Modify Linear Regression to help with Classification?????

18.6.3 Linear Classifiers w/ Hard Threshold

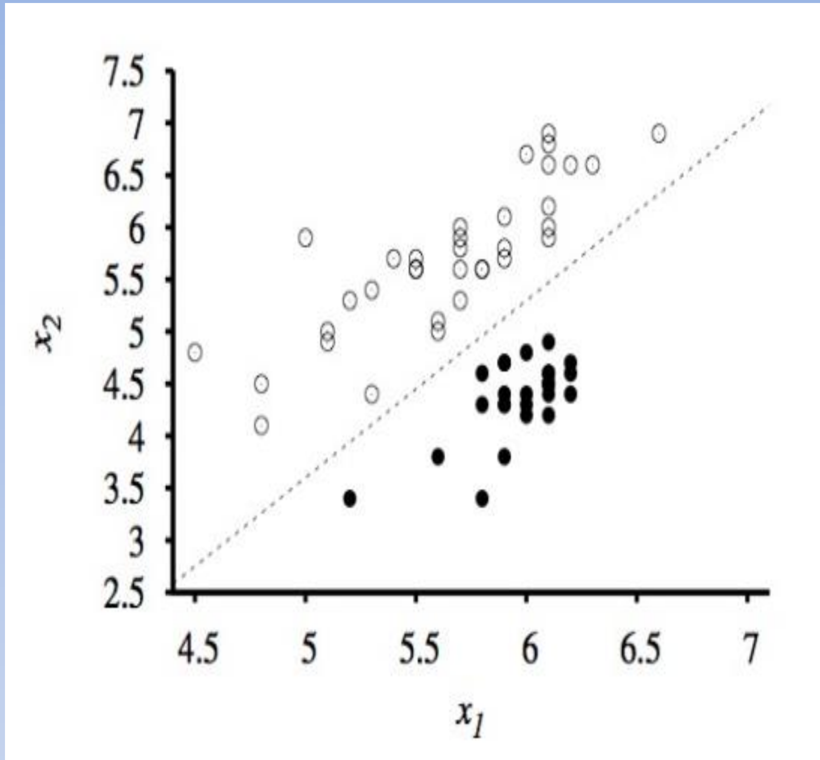


Decision Boundary:

$$-4.9 + 1.7x_1 - x_2$$

- Here we have seismic data
- Need to determine: Earthquake or Explosion
- Decision Boundary is line (or surface in higher dimensions)
- Decision Boundary separates two classes

18.6.3 Linear Classifiers w/ Hard Threshold



- *Decision Boundary:*

$$-4.9 + 1.7x_1 - x_2 = 0$$

- Explosions:

$$-4.9 + 1.7x_1 - x_2 > 0$$

- Earthquakes:

$$-4.9 + 1.7x_1 - x_2 < 0$$

Threshold Function

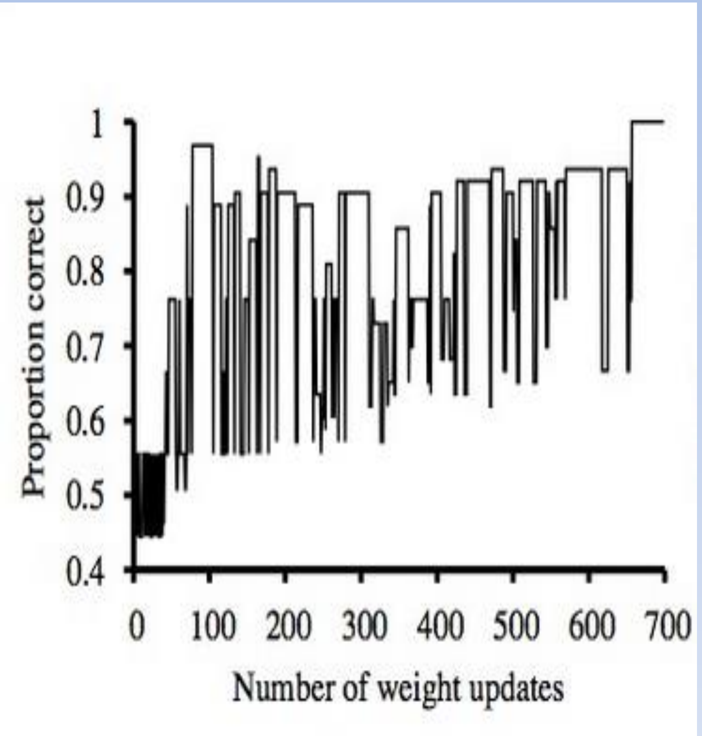
- $h_w(\boldsymbol{x}) = Threshold(\boldsymbol{w} \cdot \boldsymbol{x})$
  - $Threshold(z) = 1, if\ (z \geq 0)\ else\ 0$

- Now we need to learn these weights!
- Unfortunately, the gradient does not have the nice properties from Linear Regression!
- Fortunately, a simple rule works!

Perceptron Learning Rule

$$w_i \leftarrow w_i + \alpha \left(y - h_{\mathbf{w}}(\mathbf{x})\right) \times x_i \qquad (18.7)$$

- Now detour to Perceptrons!



The perceptron convergence procedure: Training binary output neurons as classifiers

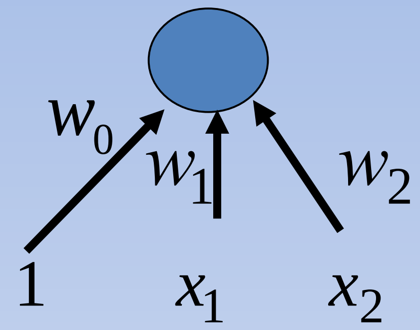
- Add an extra component with value 1 to each input vector.
  - The "bias" weight on this component is minus the threshold.
  - Now we can forget the threshold.
- Pick training cases using any policy that ensures all get picked
  - If the output unit is correct, leave its weights alone.
  - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
  - If the output unit incorrectly outputs a 1, subtract the input vector from the weight vector.
- Guaranteed to find weights getting right answer for all the training cases if any such set exists.

$$w_i \leftarrow w_i + \alpha\left(y - h_{\mathbf{w}}(\mathbf{x})\right) \times x_i \tag{18.7}$$

Learn Some Perceptrons

A	B	A or B
0	0	0
1	0	1
0	1	1
1	1	1

Learn The Weights



$$w0 + w_1 * A + w_2 * B \geq 0, \qquad True$$
$$w0 + w_1 * A + w_2 * B < 0, \qquad False$$

Question 18.6

18.6 Consider the following data set comprised of three binary input attributes (A_1, A_2, and A_3) and one binary output:

Example	A_1	A_2	A_3	Output y
$\mathbf{x}_1$	1	0	0	0
$\mathbf{x}_2$	1	0	1	0
$\mathbf{x}_3$	0	1	0	0
$\mathbf{x}_4$	1	1	1	1
$\mathbf{x}_5$	1	1	0	1

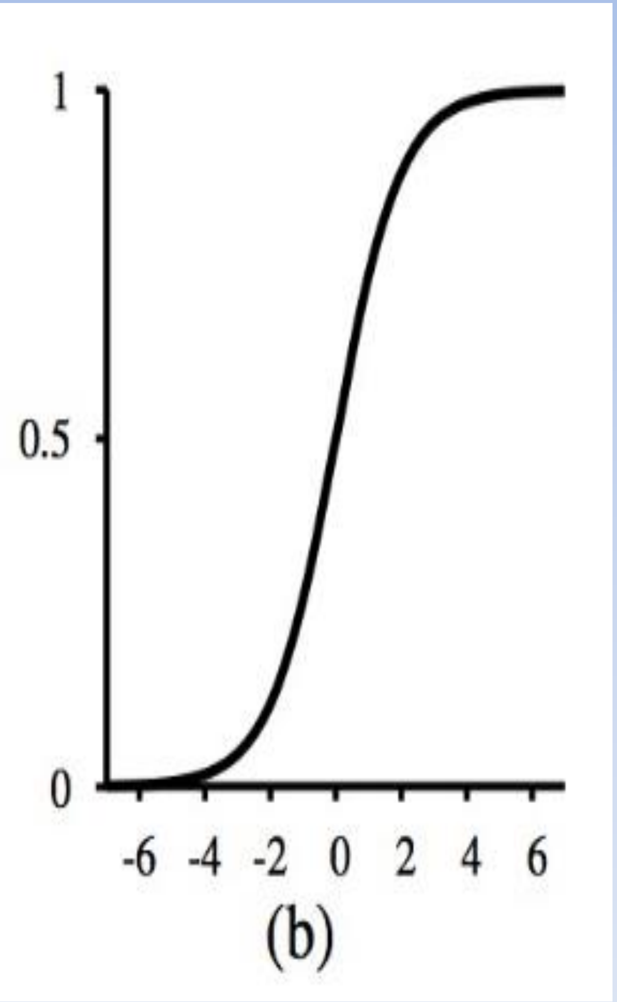
Use the algorithm in Figure 18.5 (page 702) to learn a decision tree for these data. Show the computations made to determine the attribute to split at each node.

- How about Perceptron?

18.6.4 : Improving Threshold

- Hard Threshold had issues
  - Gradient not well behaved
- Introduce a Soft Threshold
- Logistic Function:

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



Logistic Function w/ Calculus

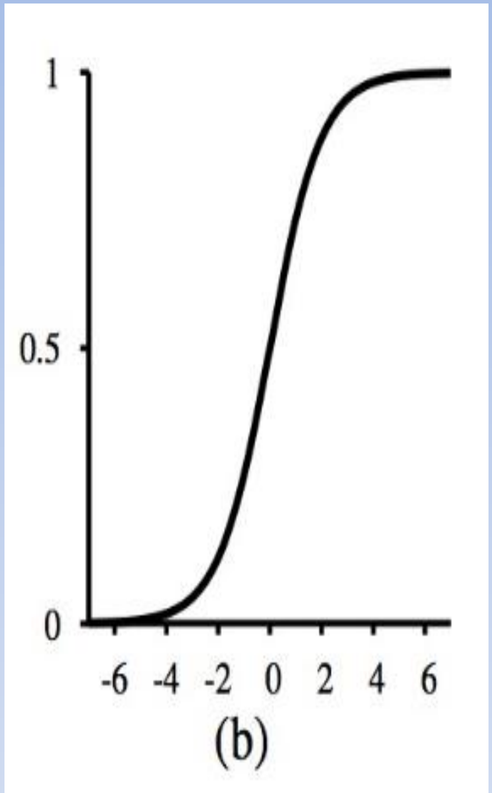
- Derivative of the Logistic Function

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

$$g'(z) = g(z)(1 - g(z))$$

Logistic Regression

$$h_w(x) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$



- View as the probability of Class = 1
- If $h_w(x) \geq 0.5$,
  - $predict\ 1,$
  - $else\ Predict\ 0$

Gradient w/ Logistic Regression Russell & Norvig 18.6

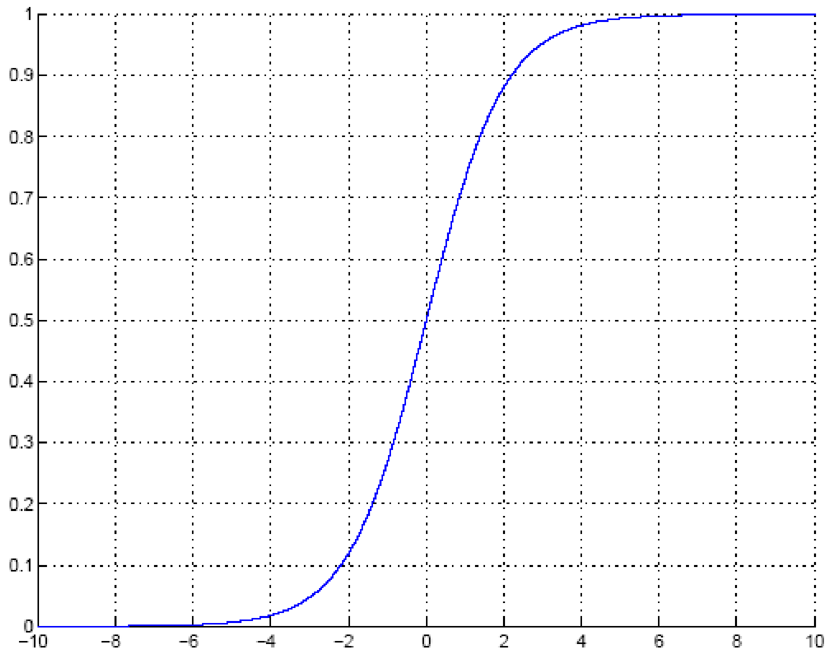
$$
\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w}\cdot\mathbf{x}) \times \frac{\partial}{\partial w_i}\mathbf{w}\cdot\mathbf{x} \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w}\cdot\mathbf{x}) \times x_i \,.
\end{aligned}
$$

$$
w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times \underline{h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))} \times x_i \,. \qquad (18.8)
$$

Another Option

- Instead of Squared Error use Cross-Entropy

Sigmoid (Logistic) Function



Calculate $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ and choose C_1 if $g(\mathbf{x}) > 0$, or

Calculate $y = \text{sigmoid}(\mathbf{w}^T\mathbf{x} + w_0)$ and choose C_1 if $y > 0.5$

Logistic Discrimination

Two classes: Assume log likelihood ratio is linear

$$\log \frac{p(\mathbf{x} \mid C_1)}{p(\mathbf{x} \mid C_2)} = \mathbf{w}^T \mathbf{x} + w_0^o$$

$$\operatorname{logit}(P(C_1 \mid \mathbf{x})) = \log \frac{P(C_1 \mid \mathbf{x})}{1 - P(C_1 \mid \mathbf{x})} = \log \frac{p(\mathbf{x} \mid C_1)}{p(\mathbf{x} \mid C_2)} + \log \frac{P(C_1)}{P(C_2)}$$

$$= \mathbf{w}^T \mathbf{x} + w_0$$

$$\text{where } w_0 = w_0^o + \log \frac{P(C_1)}{P(C_2)}$$

$$y = \hat{P}(C_1 \mid \mathbf{x}) = \frac{1}{1 + \exp\left[-\left(\mathbf{w}^T \mathbf{x} + w_0\right)\right]}$$

Training: Two Classes

$$\mathcal{X} = \{\mathbf{x}^t, r^t\}_t \quad r^t \mid \mathbf{x}^t \sim \text{Bernoulli}(y^t)$$

$$y = P(C_1 \mid \mathbf{x}) = \frac{1}{1 + \exp\left[-\left(\mathbf{w}^T\mathbf{x} + w_0\right)\right]}$$

$$l(\mathbf{w}, w_0 \mid \mathcal{X}) = \prod_t \left(y^t\right)^{\left(r^t\right)} \left(1 - y^t\right)^{\left(1 - r^t\right)}$$

$$E = -\log l$$

$$E(\mathbf{w}, w_0 \mid \mathcal{X}) = -\sum_t r^t \log y^t + \left(1 - r^t\right) \log\left(1 - y^t\right)$$

Cross-Entropy

Training: Gradient-Descent

$$E(\mathbf{w},w_0|\mathcal{X}) = -\sum_t r^t \log y^t + (1-r^t)\log(1-y^t)$$

$$\text{If } y = \text{sigmoid}(a) \quad \frac{dy}{da} = y(1-y)$$

$$\Delta w_j = -\eta\frac{\partial E}{\partial w_j} = \eta\sum_t\left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}\right)y^t(1-y^t)x_j^t$$

$$= \eta\sum_t (r^t - y^t)x_j^t, j = 1,\ldots,d$$

$$\Delta w_0 = -\eta\frac{\partial E}{\partial w_0} = \eta\sum_t(r^t - y^t)$$

Derivative Simpler

For $j = 0, \ldots, d$

    $w_j \leftarrow$rand(-0.01,0.01)

Repeat

    For $j = 0, \ldots, d$

        $\Delta w_j \leftarrow 0$

    For $t = 1, \ldots, N$

        $o \leftarrow 0$

        For $j = 0, \ldots, d$

            $o \leftarrow o + w_j x_j^t$

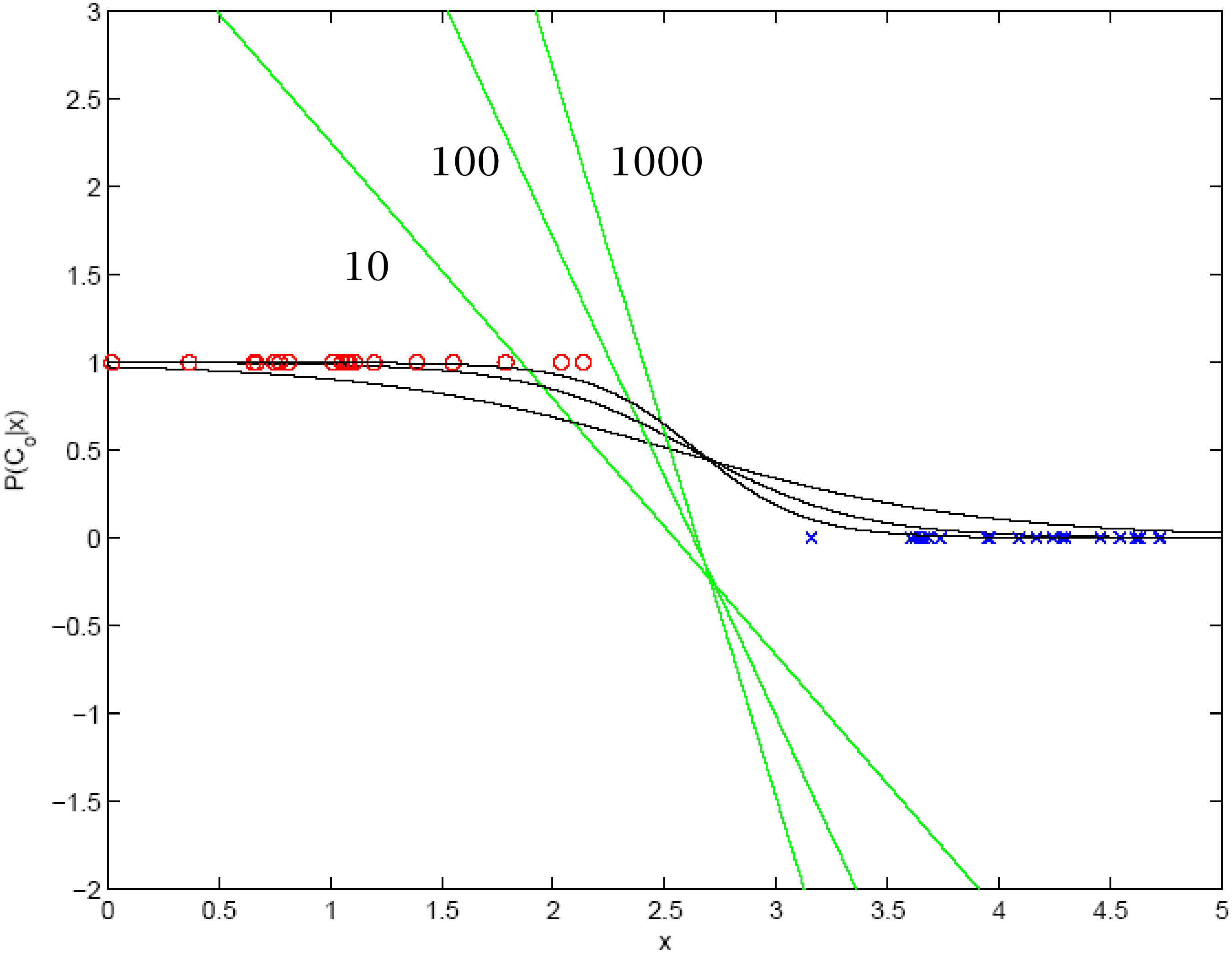
        $y \leftarrow \mathrm{sigmoid}(o)$

        $\Delta w_j \leftarrow \Delta w_j + (r^t - y) x_j^t$

    For $j = 0, \ldots, d$

        $w_j \leftarrow w_j + \eta \Delta w_j$

Until convergence



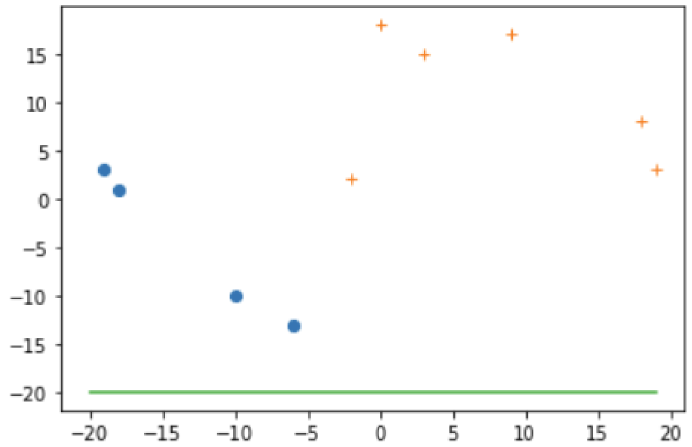
100
1000
10
P(C_o|x)
x

Logistic Regression Example

Learning Rate = 1.0

- **(x), y, H(x), y-H(x)**
- (18, 8) 1.0 1.0 0.0
- (-6, -13) 0.0 0.999 -0.999
- (0, 18) 1.0 1.0 0.0
- (-2, 2) 1.0 1.0 0.0
- (-10, -10) 0.0 1.0 -1.0
- (3, 15) 1.0 1.0 0.0
- (19, 3) 1.0 1.0 0.0
- (-18, 1) 0.0 1.0 -1.0
- (-19, 3) 0.0 1.0 -1.0
- (9, 17) 1.0 1.0 0.0
- Average (y-H(x)) = -0.4

[130]




```
Count =  0
W =  (20, 0, 1)
Initial Error=  4.0
```

[126]

```
for x, y in zip(X,Y):
  print (x, y, round(Hgx(w,x),3), round(y-Hgx(w,x), 3) )
```

```
(18, 8) 1.0 1.0 0.0
(-6, -13) 0.0 0.999 -0.999
(0, 18) 1.0 1.0 0.0
(-2, 2) 1.0 1.0 0.0
(-10, -10) 0.0 1.0 -1.0
(3, 15) 1.0 1.0 0.0
(19, 3) 1.0 1.0 0.0
(-18, 1) 0.0 1.0 -1.0
(-19, 3) 0.0 1.0 -1.0
(9, 17) 1.0 1.0 0.0
```

Learning Rate = 1.0

- **(x), y, H(x), x1*(y-H(x))**
- (18, 8) 1.0 1.0 0.0
- (-6, -13) 0.0 0.999 5.995
- (0, 18) 1.0 1.0 0.0
- (-2, 2) 1.0 1.0 -0.0
- (-10, -10) 0.0 1.0 10.0
- (3, 15) 1.0 1.0 0.0
- (19, 3) 1.0 1.0 0.0
- (-18, 1) 0.0 1.0 18.0
- (-19, 3) 0.0 1.0 19.0
- (9, 17) 1.0 1.0 0.0
- Average (y-H(x))*x1 = 5.3

Learning Rate = 1.0

- **(x), y, H(x), x2*(y-H(x))**
- (18, 8) 1.0 1.0 0.0
- (-6, -13) 0.0 0.999 12.988
- (0, 18) 1.0 1.0 0.0
- (-2, 2) 1.0 1.0 0.0
- (-10, -10) 0.0 1.0 10.0
- (3, 15) 1.0 1.0 0.0
- (19, 3) 1.0 1.0 0.0
- (-18, 1) 0.0 1.0 -1.0
- (-19, 3) 0.0 1.0 -3.0
- (9, 17) 1.0 1.0 0.0
- Average (y-H(x))*x2 = 1.9

New Weights


```
-0.4 5.3 1.9
Count =  1
W =  (19.600095645030624, 5.2994079699952525, 2.8987702357723175)
Errors=  3.9981 0.0 3.9981
```