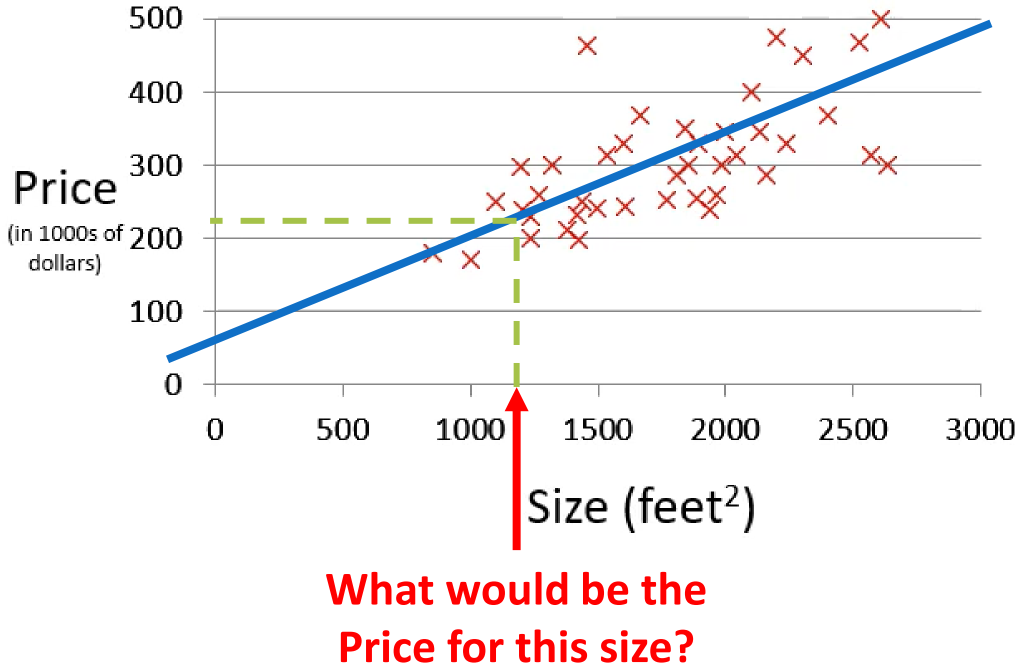
Revision

COST FUNCTION, ERROR, LOSS

LR with One Variable: Alternative Perspective

Size $(Feet^2)$	Price $\$(\times 1000)$
1500	190
2250	285
2740	420
2318	300
2500	350
1250	180
…	…

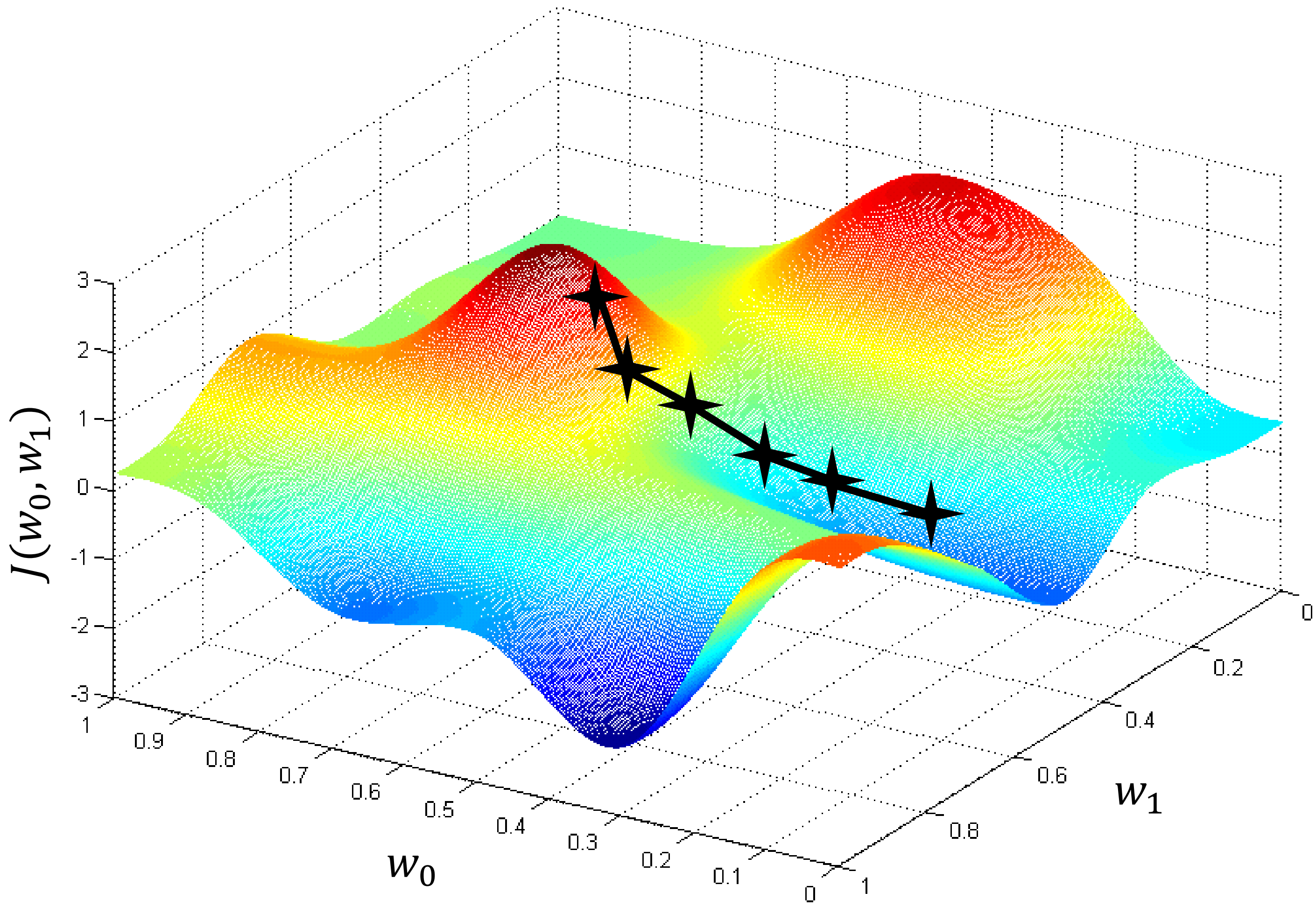


Notations:

$$m = Total\ Number\ of\ Training\ Samples$$
$$x = Feature$$
$$y = Label$$

$(x^{(i)}, y^{(i)})$: the ith sample in the dataset
i.e., when $i = 1, x^{(1)} = 2250, y^{(1)} = 285$

Gradient Descent Algorithm



Credit: Andrew Ng

Gradient Descent Algorithm

Repeat until convergence {

$$\boldsymbol{w_j := w_j - \alpha \frac{\partial}{\partial w_j}\left(J(w_0, w_1)\right)}$$

Simultaneously update $j = 0$ and $j = 1$

(for $j = 1$ and $j = 0$)

}

Correct: Simultaneous Update

$$\boldsymbol{temp0 := w_0 - \alpha \frac{\partial}{\partial w_0}\left(J(w_0, w_1)\right)}$$

$$\boldsymbol{temp1 := w_1 - \alpha \frac{\partial}{\partial w_1}\left(J(w_0, w_1)\right)}$$

$$\boldsymbol{w_0 = temp\ 0}$$

$$\boldsymbol{w_1 = temp\ 1}$$

Incorrect:

$$\boldsymbol{temp0 := w_0 - \alpha \frac{\partial}{\partial w_0}\left(J(w_0, w_1)\right)}$$

$$\boldsymbol{w_0 = temp\ 0}$$

$$\boldsymbol{temp1 := w_1 - \alpha \frac{\partial}{\partial w_1}\left(J(w_0, w_1)\right)}$$

$$\boldsymbol{w_1 = temp\ 1}$$

Gradient Descent Algorithm

Repeat until convergence {

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} \left(J(w_0, w_1) \right)$$

Simultaneously update $j = 0$ and $j = 1$

(for $j = 1$ and $j = 0$)

}

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)} \right)^2$$

It's evident as per this cost function that we calculate cost $J\ (w_0, w_1)$ for all training instances before performing the update!

Gradient Descent and Linear Regression

Gradient Descent Algorithm

Repeat until convergence {

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} \left(J(w_0, w_1)\right)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

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

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Self Study: Derivation of Gradient Descent

$$\frac{\partial}{\partial w_j}\left(J(w_0, w_1)\right) = \frac{\partial}{\partial w_j}\frac{1}{2m}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$\frac{\partial}{\partial w_j}\left(J(w_0, w_1)\right) = \frac{\partial}{\partial w_j}\frac{1}{2m}\sum_{i=1}^{m}\left(w_0 + w_1 x^{(i)} - y^{(i)}\right)^2$$

$$j = 0 : \frac{\partial}{\partial w_0}\left(J(w_0, w_1)\right) = \frac{1}{m}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$j = 0 : \frac{\partial}{\partial w_0}\left(J(w_0, w_1)\right) = \frac{1}{m}\sum_{i=1}^{m}\left(w_0 + w_1 x^{(i)} - y^{(i)}\right)^2$$

$$j = 1 : \frac{\partial}{\partial w_1}\left(J(w_0, w_1)\right) = \frac{1}{m}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 x^{(i)}$$

$$j = 1 : \frac{\partial}{\partial w_1}\left(J(w_0, w_1)\right) = \frac{1}{m}\sum_{i=1}^{m}\left(w_0 + w_1 x^{(i)} - y^{(i)}\right)^2 x^{(i)}$$

Gradient Descent Algorithm

Repeat until convergence {

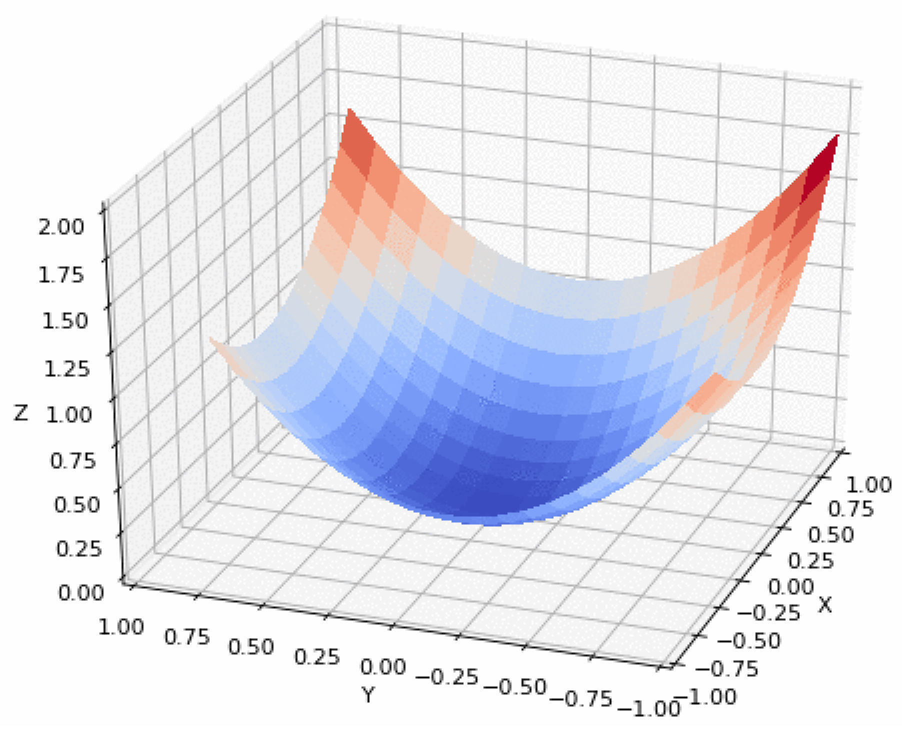
$$\boldsymbol{w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2}$$

$$\boldsymbol{w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 . \, x^{(i)}}$$

}

Update w_0 and w_1 simultaneously.

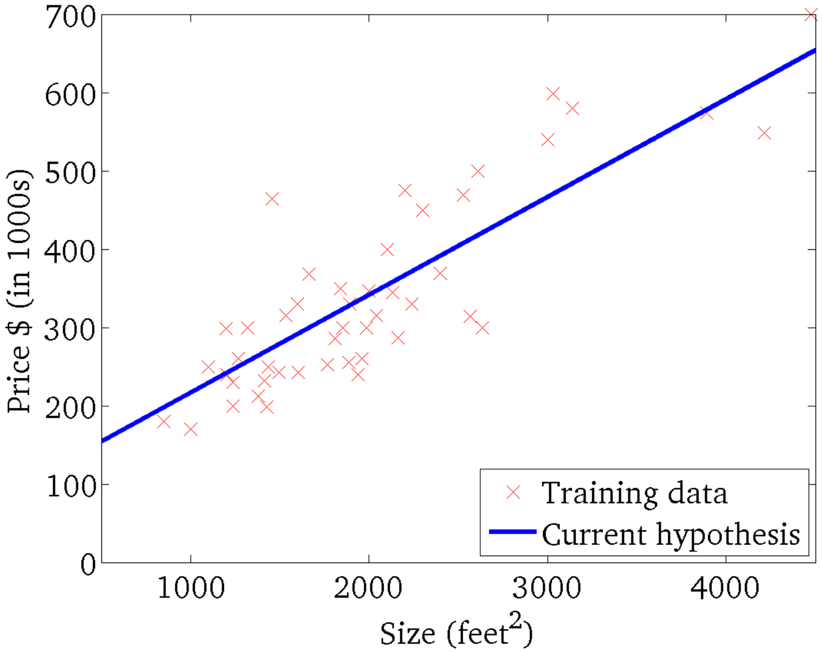
Convex Function



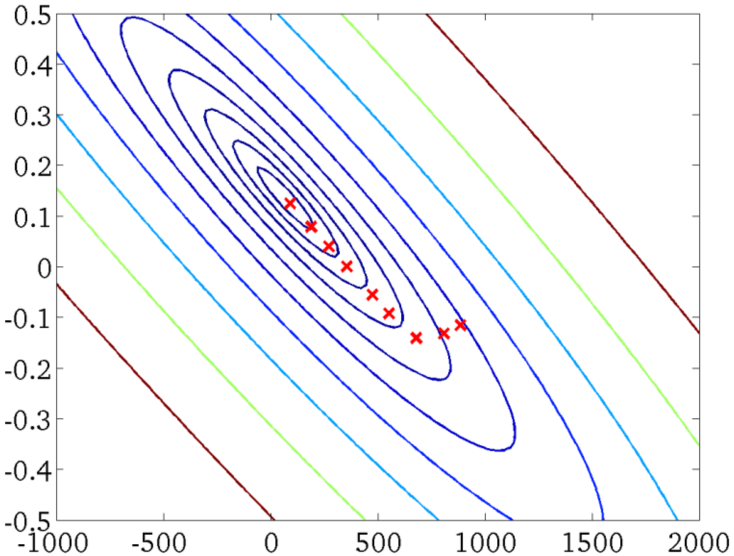
Implementation Tip: Stop the weights update if error does not reduce for k iterations/steps.

Credit: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

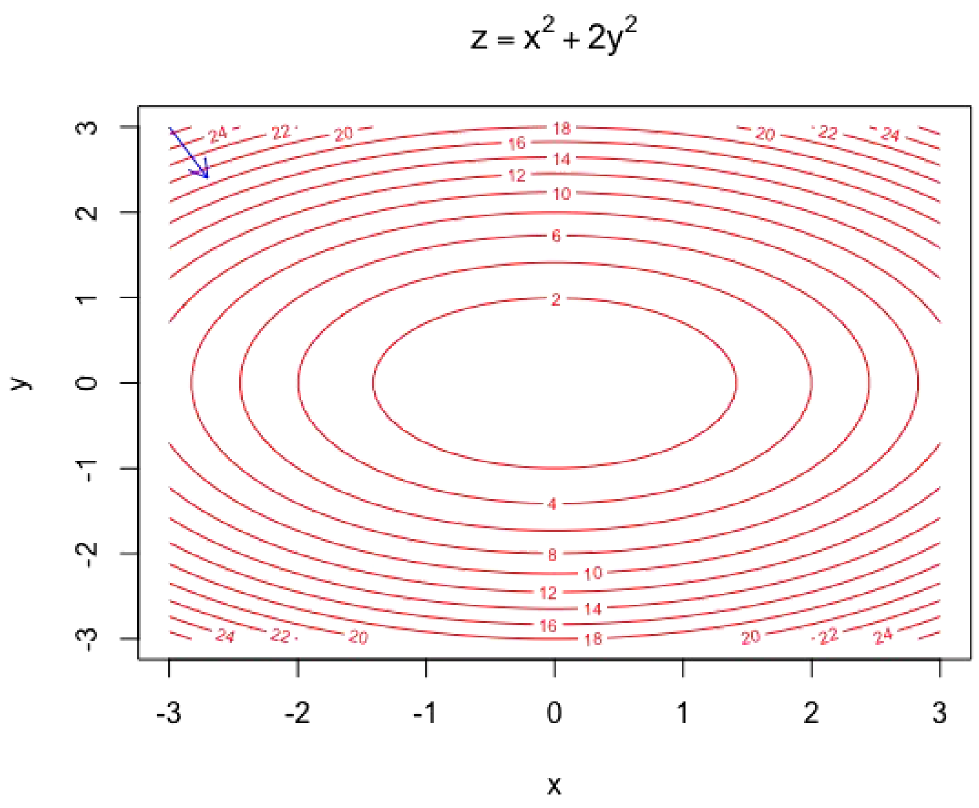
$$h(x) = w_0 + w_1$$



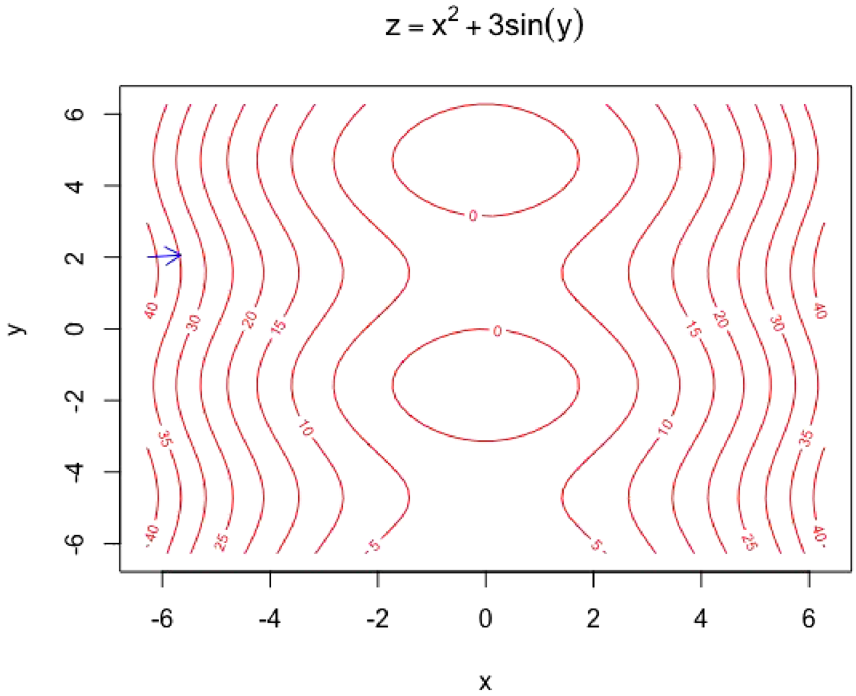
$$J(w_0, w_1)$$



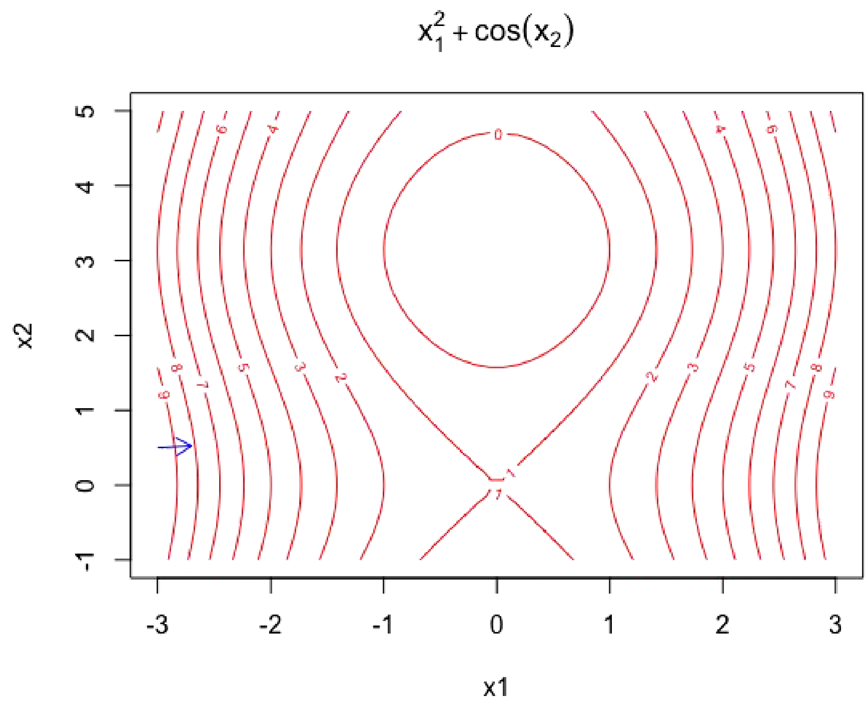
Contour Plots: Alternative Form



Contour Plots: Alternative Form



Contour Plots: Alternative Form



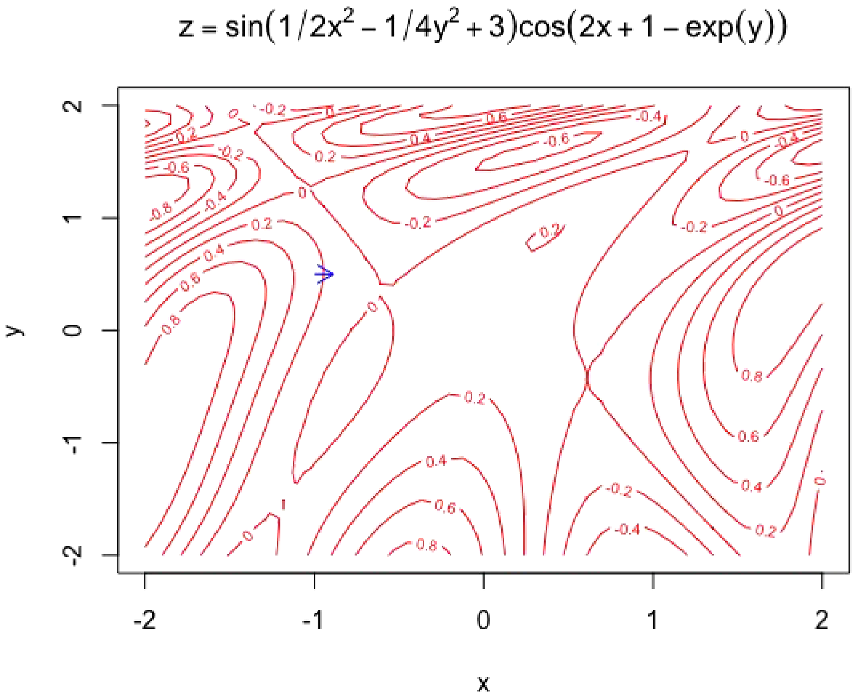
Point to Remember…

- **Features that produce nice circles on error surface cause quicker convergence**

- **Features that produce irregular shapes on error surface cause slower convergence**

- **We try to normalize features in a way that every feature have same scale.**

Contour Plots: Alternative Form



Animation Credit: https://yihui.org/animation/example/grad-desc/

Gradient Descent Types

❑ Batch Gradient Descent

- Each step of gradient uses all the training examples.
- Computes the gradient of the cost function with respect to the parameters for the entire training dataset
- What if we have a very large training dataset?

❑ Stochastic Gradient Descent

- Performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$
- Cost function plot will show erratic movement (Drunken Walk)
- Converges quickly as compared to BGD

❑ Mini-Batch Gradient Descent

- Takes the best of the two worlds
- Perform an update for every mini-batch of n examples.

Useful Resource: https://ruder.io/optimizing-gradient-descent/

Multivariate Linear Regression

Multiple Variables (Features)

Size $(Feet^2)$	Price \$$(\times 1000)$
1500	190
2250	285
2740	420
2318	300
2500	350
1250	180
…	…

$$\boldsymbol{h(x) = w_0 + w_1 x}$$

Notations:

$$m = Total\ Number\ of\ Training\ Samples$$
$$x = Feature$$
$$y = Label$$

$(x^{(i)}, y^{(i)})$: the ith sample in the dataset
i.e., when $i = 1, x^{(1)} = 2250, y^{(1)} = 285$

Multiple Variables (Features)

Size $(Feet^2)$	# of Bedrooms	# of Floors	Building Age (years)	Price $\$(\times 1000)$
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

$$h(X) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Notations:

$$m = Total\ Number\ of\ Training\ Samples$$
$$n = Total\ Number\ of\ Features$$
$$x = Feature$$
$$y = Label$$

$(X_j^{(i)}, y_j^{(i)}$: the jth feature of the ith sample in the dataset

Hypothesis

- **Previously:**

$$\boxed{\boldsymbol{h(x) = w_0 + w_1 x}}$$

- **Now:**

$$\boxed{\boldsymbol{h(X) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}}$$

Where $X = [x_1, x_2, \dots, x_n]$ is the n −dimensional feature vector, and $W = [w_0, w_1, \dots, w_n]$ is an $n + 1$ dimensional vector of weights.

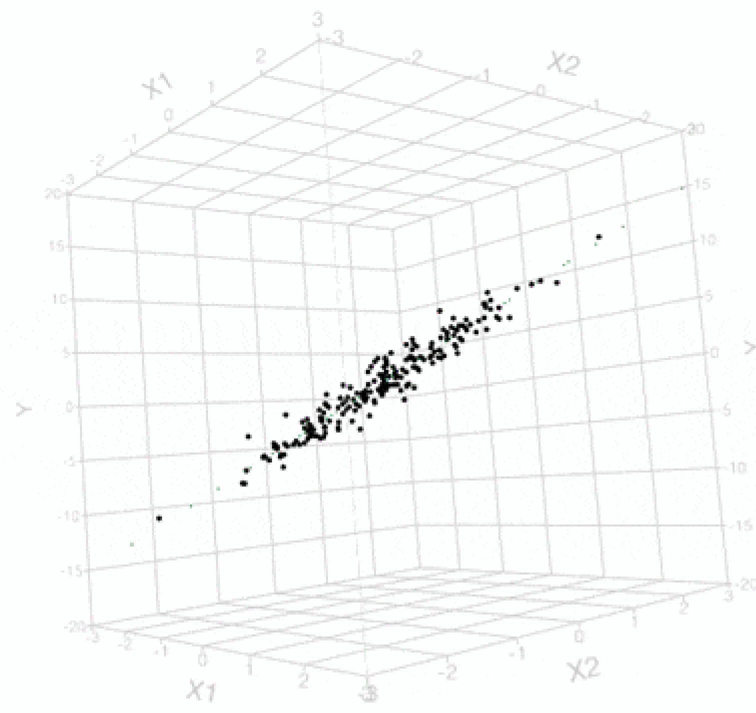
$$\boxed{\boldsymbol{X \in \mathbb{R}^n, W \in \mathbb{R}^{n+1}}}$$

Geometric Interpretation

$$\boldsymbol{h(X) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}$$

Where $X = [x_1, x_2, \ldots, x_n]$ is the n −dimensional feature vector, and $W = [w_0, w_1, \ldots, w_n]$ is an $n + 1$ dimensional vector of weights.

$\boldsymbol{h(X) = w_0 + w_1 x_1 + w_2 x_2}$ **(a 2-D hyperplane in 3-D space)**



Credit: https://www.jmp.com/en_us/statistics-knowledge-portal/what-is-multiple-regression/fitting-multiple-regression-model.html

Hypothesis

$$\boldsymbol{h(X) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n}$$

Where $X = [x_1, x_2, \dots, x_n]$ is the n −dimensional feature vector, and $W = [w_0, w_1, \dots, w_n]$ is an $n + 1$ dimensional vector of weights.

To make this more uniform, assume $x_0 = 1$ to get:

$$\boldsymbol{h(X) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n}$$

Where $X = [x_0, x_1, x_2, \dots, x_n]$ is the $n + 1$ dimensional feature vector, and $W = [w_0, w_1, \dots, w_n]$ is an $n + 1$ dimensional vector of weights.

$$\boldsymbol{X \in \mathbb{R}^{n+1}, W \in \mathbb{R}^{n+1}}$$

Credit: https://www.jmp.com/en_us/statistics-knowledge-portal/what-is-multiple-regression/fitting-multiple-regression-model.html

Dataset with x_0

x_0	Size $(Feet^2)$	# of Bedrooms	# of Floors	Building Age (years)	Price $\$(\times 1000)$
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178
…	…	…	…	…	…

This is useful to compute whole hypothesis equation via matrix multiplication…

Vectorizing the Notation

$$\boxed{h(X) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n}$$

Recall that this equation can be decomposed into vectors and we can perform dot product or matrix multiplication to get the same result…

$$W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}, X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \text{ where } X \in \mathbb{R}^{n+1}, \mathbf{W} \in \mathbb{R}^{n+1}$$

$$\boxed{h(X) = W^T X}$$

$$[w_0, w_1, \dots, w_n] \times \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = [\,]$$

But this was for one record/instance only.
We need to do this for all training instances.

Why did we take transpose of W?

For m Training Instances

$$\boldsymbol{h(X) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n}$$

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_0} \\ \boldsymbol{w_1} \\ \vdots \\ \boldsymbol{w_n} \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_0^{(1)}} & \boldsymbol{x_0^{(2)}} & & \boldsymbol{x_0^{(m)}} \\ \boldsymbol{x_1^{(1)}} & \boldsymbol{x_1^{(2)}} & \dots & \boldsymbol{x_1^{(m)}} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{x_n^{(1)}} & \boldsymbol{x_n^{(2)}} & & \boldsymbol{x_n^{(m)}} \end{bmatrix}$$

Now that you have predicted and actual labels for all instances, compute cost/metrics.

Actual Labels

$$\boxed{\left[\boldsymbol{y^{(1)}, y^{(2)}, \dots, y^{(m)}}\right]}$$

$$\boldsymbol{h(X) = W^TX} = [\boldsymbol{w_0, w_1, \dots, w_n}] \times \begin{bmatrix} \boldsymbol{x_0^{(1)}} & \boldsymbol{x_0^{(2)}} & & \boldsymbol{x_0^{(m)}} \\ \boldsymbol{x_1^{(1)}} & \boldsymbol{x_1^{(2)}} & \dots & \boldsymbol{x_1^{(m)}} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{x_n^{(1)}} & \boldsymbol{x_n^{(2)}} & & \boldsymbol{x_n^{(m)}} \end{bmatrix} = \left[\, \boldsymbol{h\left(x^{(1)}\right), h\left(x^{(2)}\right), \dots, h\left(x^{(m)}\right)}\right]$$

Predictions for all instances

Summary

- **Hypothesis**

$$\boldsymbol{h(X) = W^T X = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}$$

- **Parameter Vector**

$$\boldsymbol{W = w_0, w_1, \dots, w_n}$$

- **Feature Vector**

$$\boldsymbol{X = x_0, x_1, \dots, x_n}$$

- **Cost Function:**

$$\boldsymbol{J(W) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2}$$

- **Gradient Descent**

Repeat until convergence {

$$\boldsymbol{w_j := w_j - \alpha \frac{\partial}{\partial w_j} \big(J(W)\big)}$$ *// simultaneously update for all $j = 0 \dots n$*

}

Summary: Gradient Descent

- **Previously** $(n = 1)$:

Repeat until convergence {

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 . x^{(i)}$$

}

- **Now** $(n \geq 1)$:

Repeat until convergence {

$$w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 . x_j^i$$

}

Summary: Gradient Descent

- How the equation looks like for each value of w_j

$$\boldsymbol{w_o \coloneqq w_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 . x_0^i}$$

$$\boldsymbol{w_1 \coloneqq w_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 . x_1^i}$$

$$\boldsymbol{w_2 \coloneqq w_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 . x_2^i}$$

$$\vdots$$

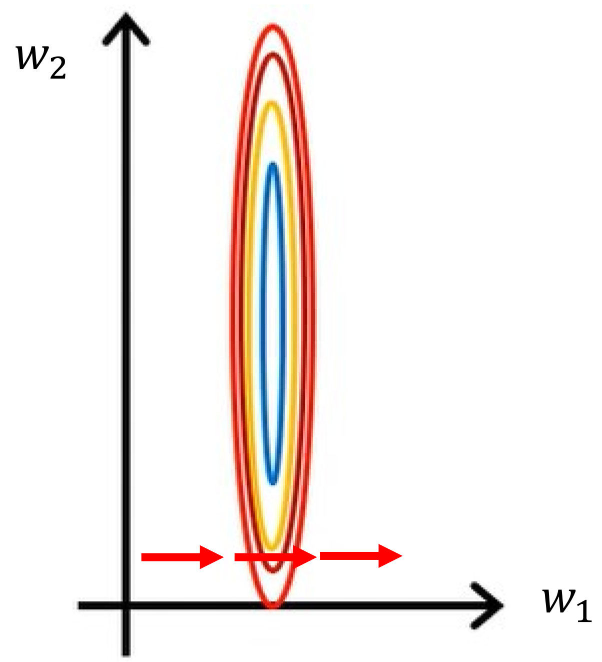
Linear Regression

PRACTICAL ISSUES

Feature Scaling

- Make sure features are on a similar scale or the learning will occur very slowly.
- As α needs to be small enough for the dimension with the smallest scale

E.g. $x_1 = \text{size } (0 - 2000 \text{ feet}^2)$
$x_2 = \text{number of bedrooms } (1 - 5)$



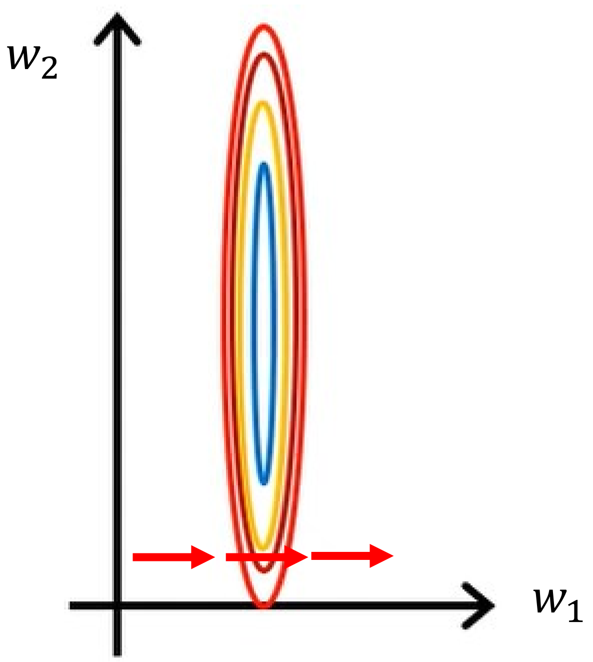
α (aka learning rate) must be too small to avoid missing the minima

This makes convergence very slow as we are taking very small steps towards minima...

Feature Scaling

- Make sure features are on a similar scale or the learning will occur very slowly.
- As α needs to be small enough for the dimension with the smallest scale

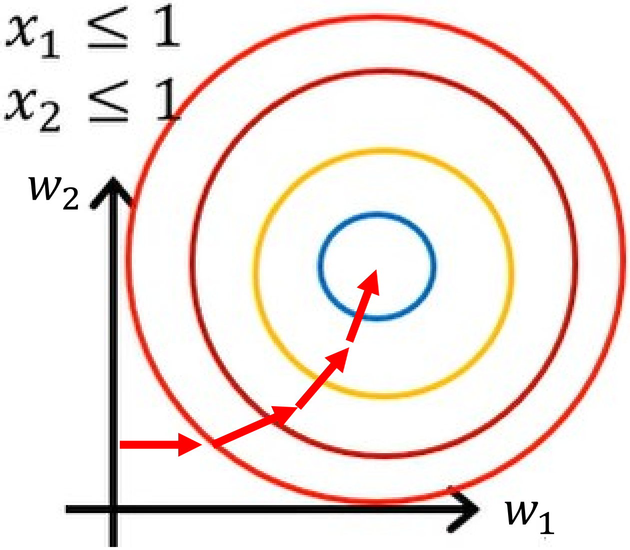
E.g. $x_1 = \text{size } (0 - 2000 \text{ feet}^2)$
$x_2 = \text{number of bedrooms } (1 - 5)$



E.g. $x_1 = \dfrac{\text{size } (\text{feet}^2)}{2000}$

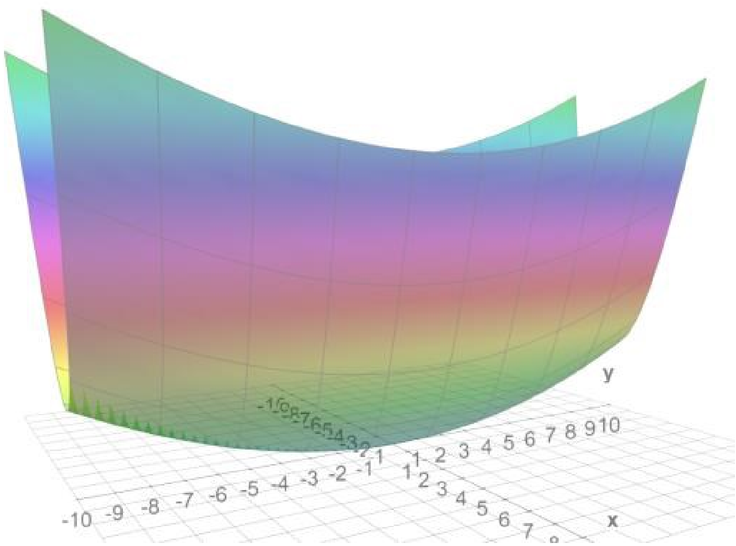
$x_2 = \dfrac{\text{number of bedrooms}}{5}$

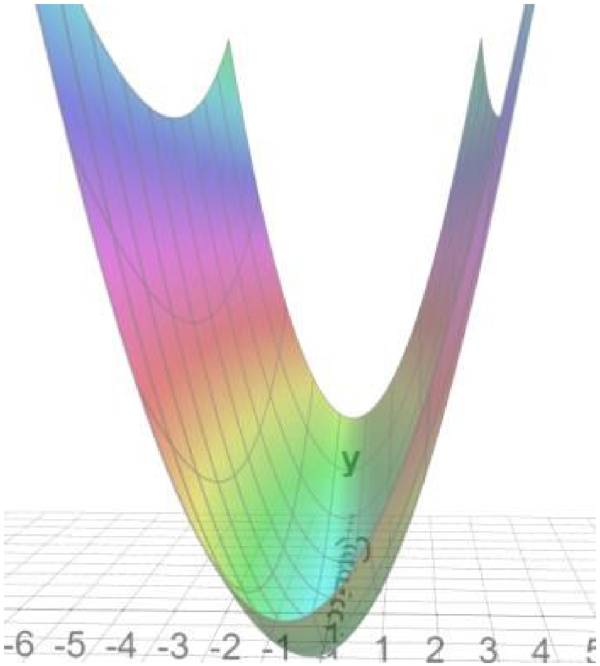
$0 \le x_1 \le 1$
$0 \le x_2 \le 1$



Ravines

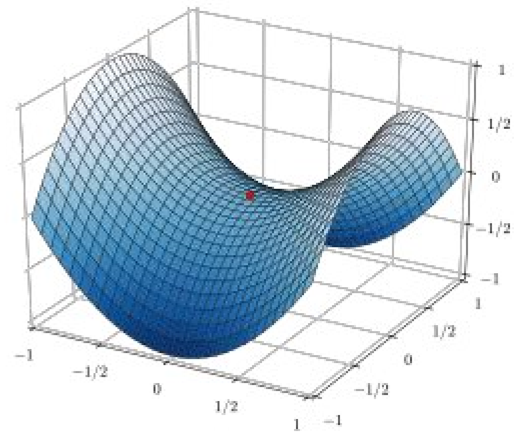
- SGD has trouble navigating ravines, i.e, areas where the surface curves much more steeply in one dimension than in another, which are common around local minima

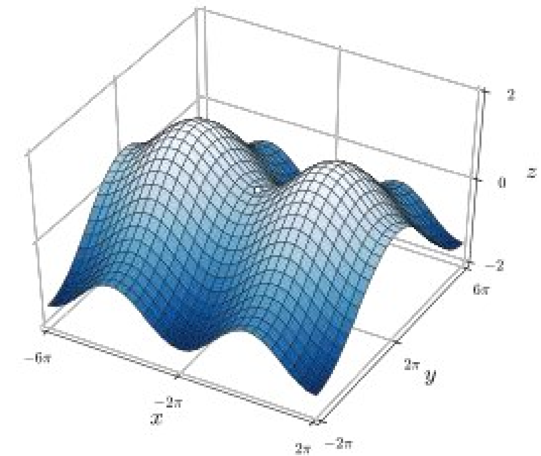


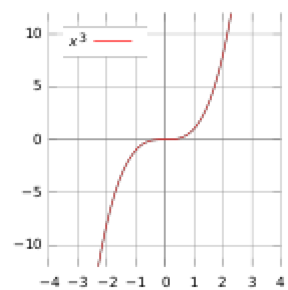


Saddle Points

- A challenge of minimizing highly non-convex error function (common for neural networks), is avoiding getting trapped in their numerous suboptimal local minima.
- The difficulty arises not from local minima, but from saddle points (i.e., the points where one dimension slopes up and another sloes down).
- These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.







Solution: Add momentum to the SGD equation.

Mean Normalization

- Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$)
- Suppose: $\mu_1 = 1000, \mu_2 = 2$

E.g. $\boldsymbol{x_1} = \dfrac{\textbf{size (feet}^2\textbf{)} - \mathbf{1000}}{\mathbf{2000}}$

$\boldsymbol{x_2} = \dfrac{\textbf{number of bedrooms} - \mathbf{2}}{\mathbf{5}}$

- **Ideally:**

$$\boldsymbol{x_1} = \frac{\boldsymbol{x_1 - \mu_1}}{\boldsymbol{S_1}}, \boldsymbol{x_1} = \frac{\boldsymbol{x_2 - \mu_2}}{\boldsymbol{S_2}}$$

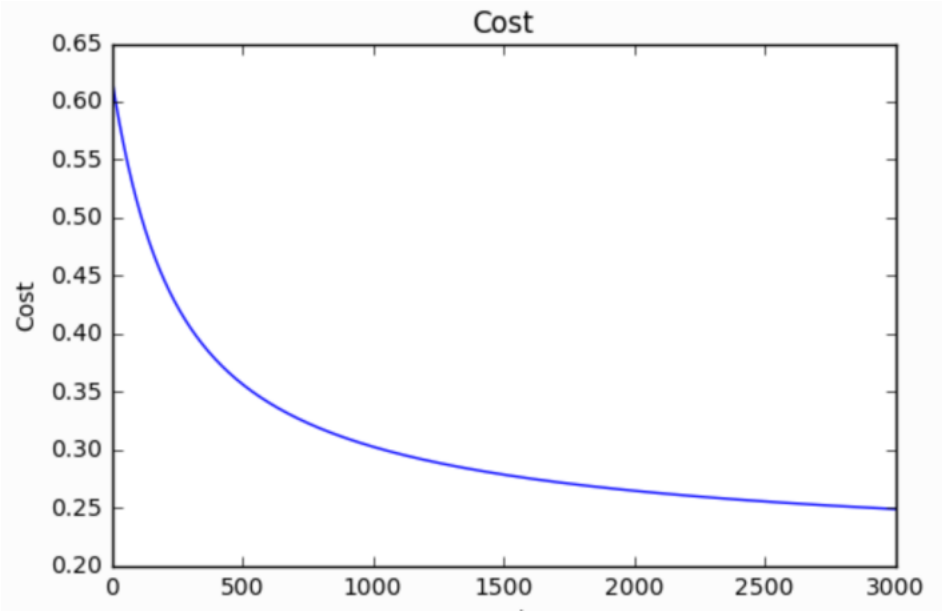
Implementation Tips:

(1) You can use sklearn "StandardScalar" method for this.

(2) You can define a pipeline with sklearn that conveniently does this preprocessing first and then apply classifier/regressor.

Debugging

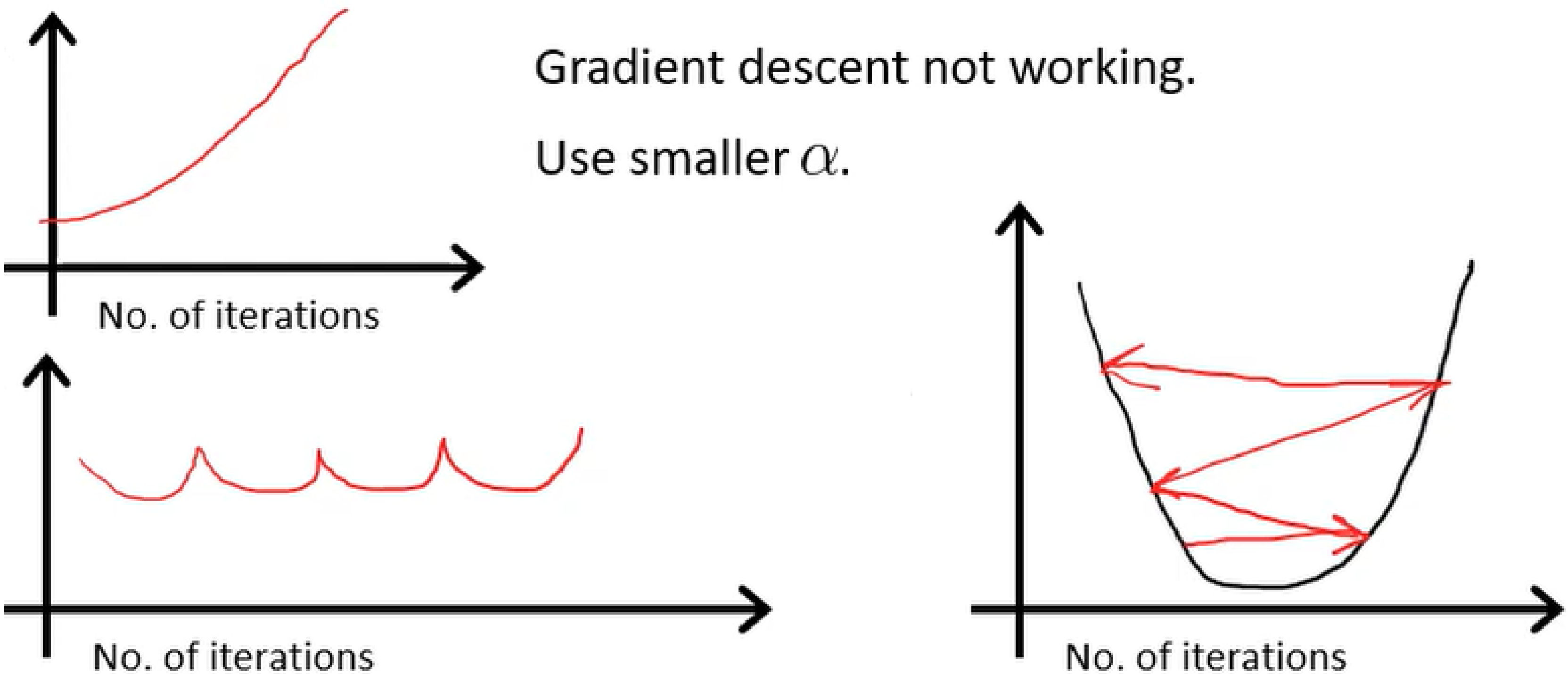
- How to make sure gradient descent is working correctly?
- How to choose the learning rate α?
- Ideally, cost should decrease after each iteration
  - Reduce learning rate
  - Declare convergence if $J(W)$ decreases by less than 10^{-3}



Iteration

Debugging

- For sufficiently small α, $J(W)$ should decrease on every iteration
- Warning: Too small α can cause gradient descent to converge slowly



Summary

- If α is too small: slow convergence
- If α is too large: $J(\theta)$ may not decrease on every iteration
- To choose α, try:
  - …, 0.001, 0.01, 0.1, 1, …

Book Reading

- Murphy – Chapter 1, Chapter 14
- Tom Mitchel (TM) – Chapter 4