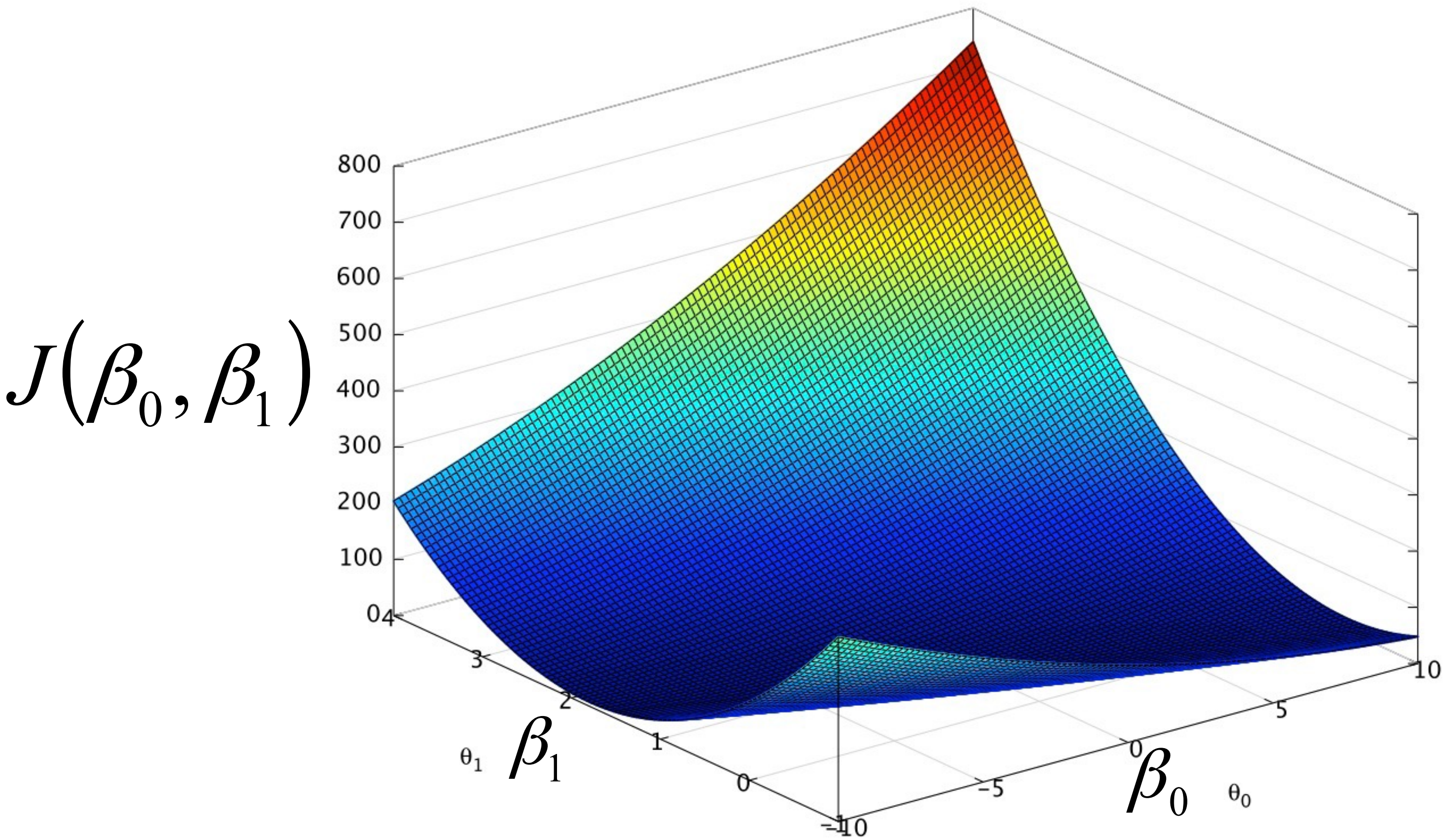
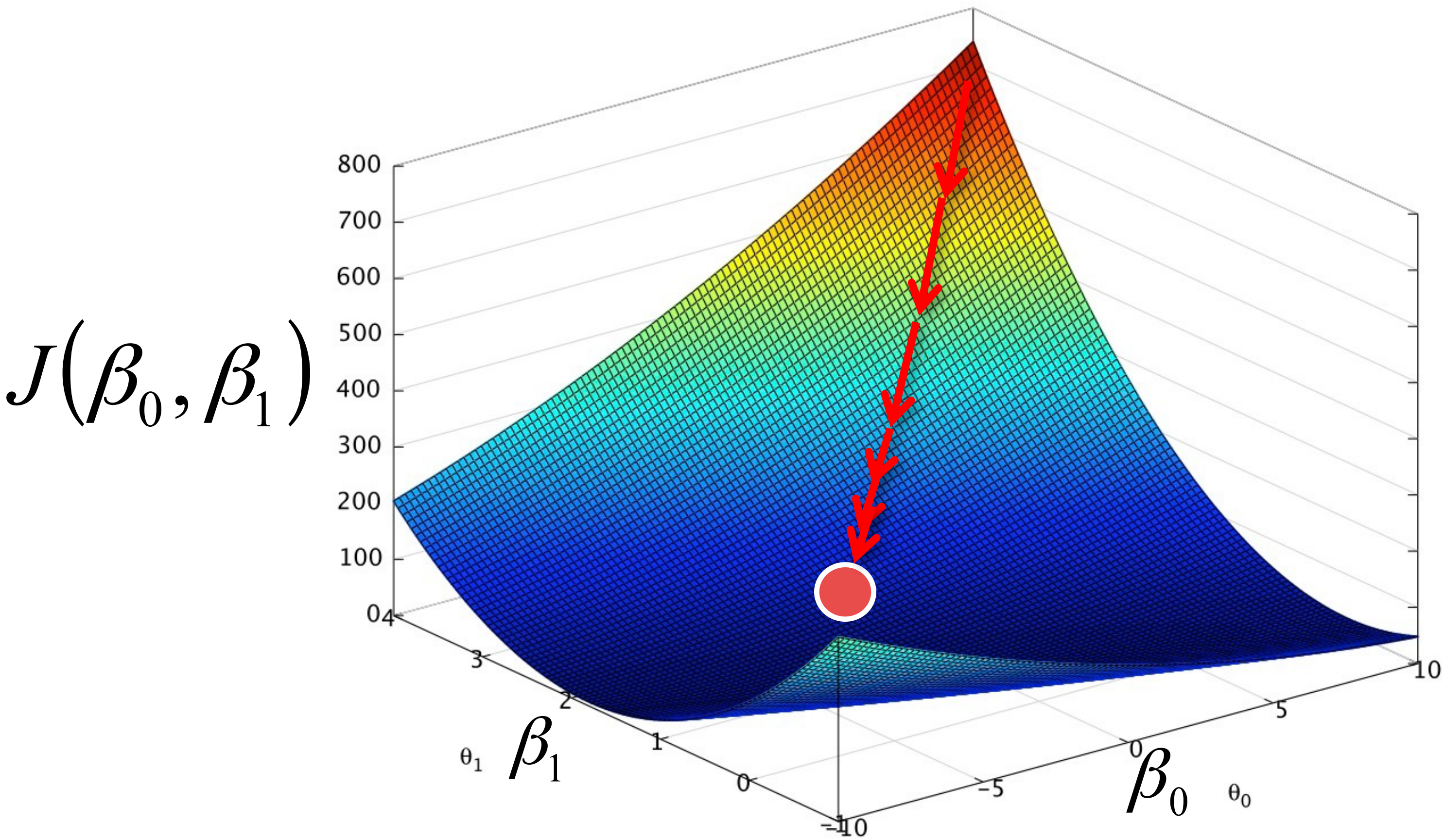
Stochastic Gradient Descent

Gradient Descent

Start with a cost function J(β):

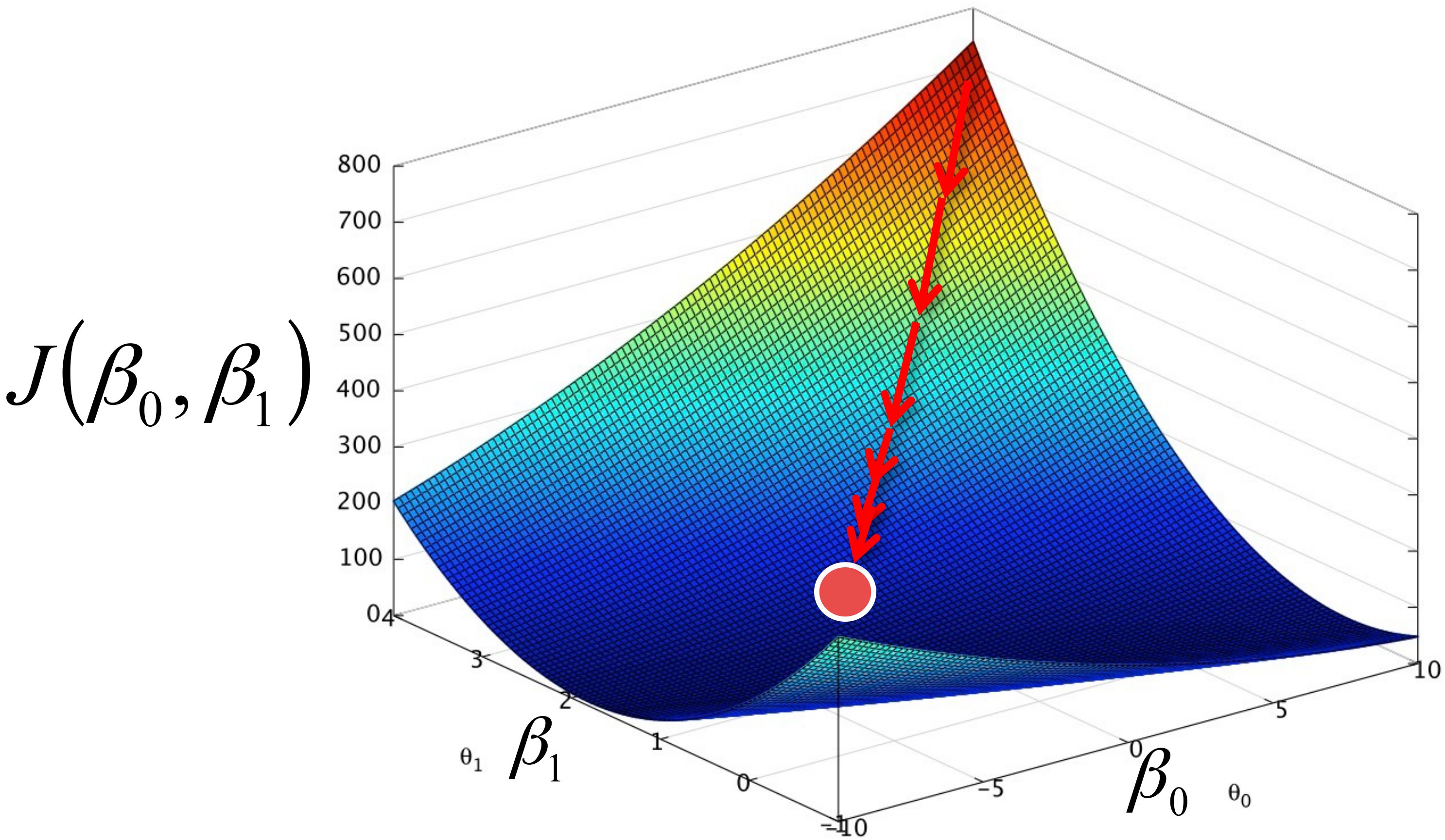


Gradient Descent



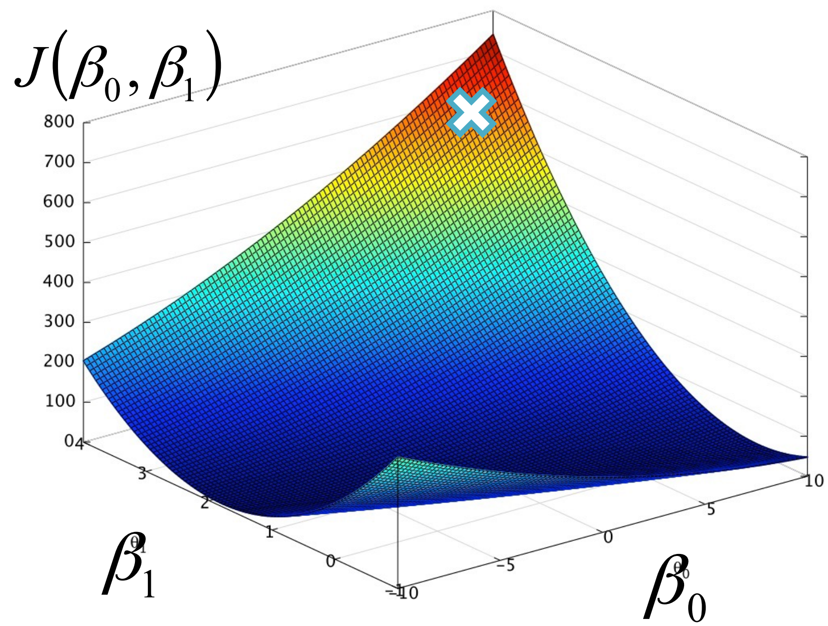
Gradient Descent

Then gradually move to the minimum.



Gradient Descent with Linear Regression

How can we do this?

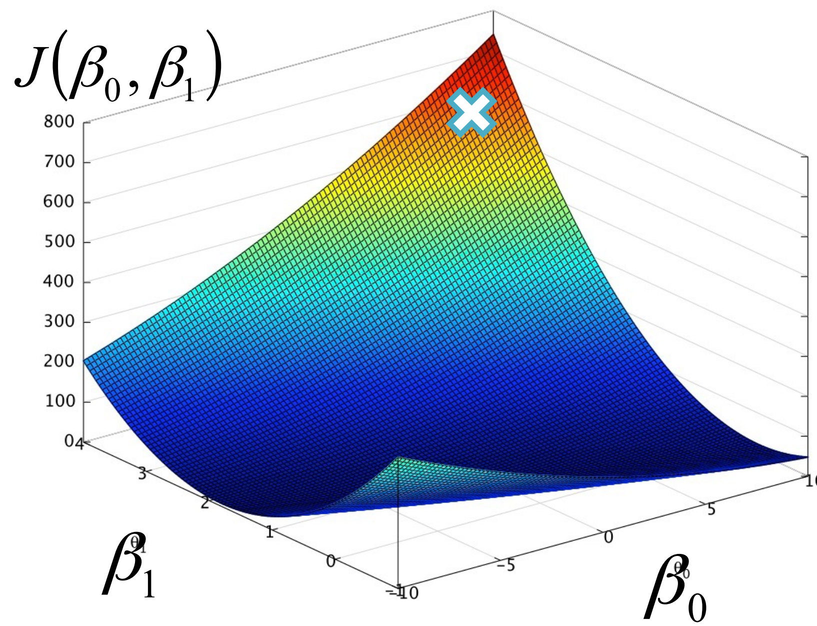


Gradient Descent with Linear Regression

How can we do this?

(without seeing the graph of J(β)!)

Start with the function J(β):

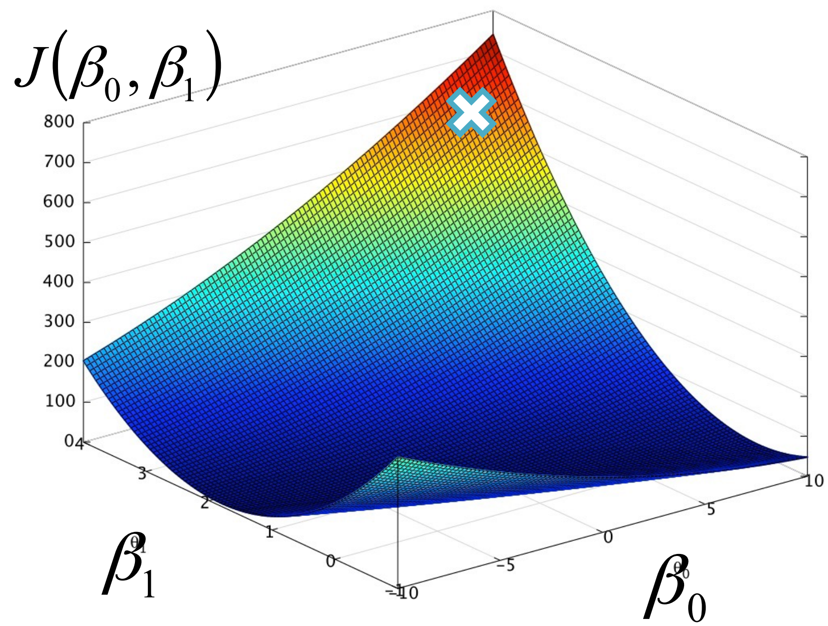


Gradient Descent with Linear Regression

How can we do this?
(without seeing the graph of J(β)!)

Start with the function J(β):

$$J\left(\beta_0,\beta_1\right) = 1/2\sum_{i=1}^{m}\left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2$$



Gradient Descent with Linear Regression

How can we do this?

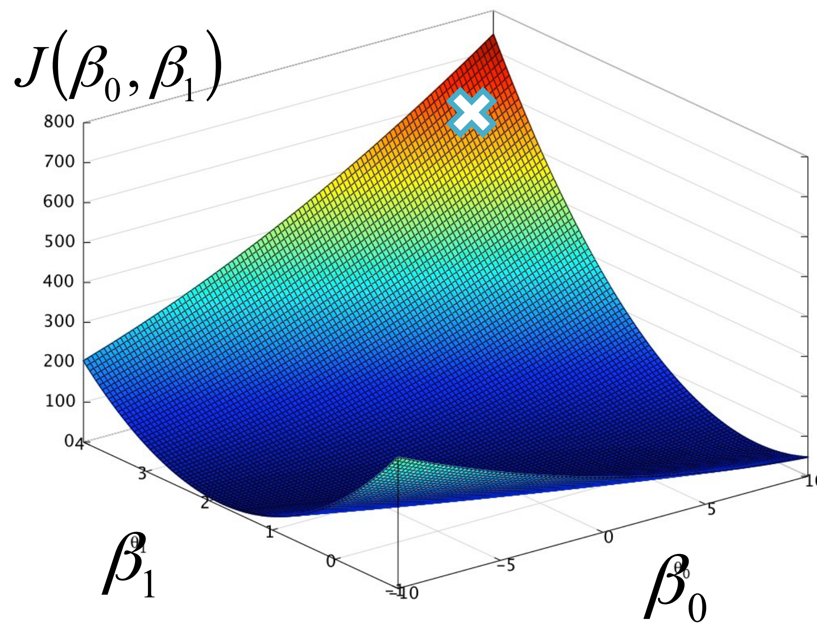
(without seeing the graph of J(β)!)

Start with the function J(β):

$$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

and compute its

<u>gradient vector</u> $\nabla J(\beta)$.



Gradient Descent with Linear Regression

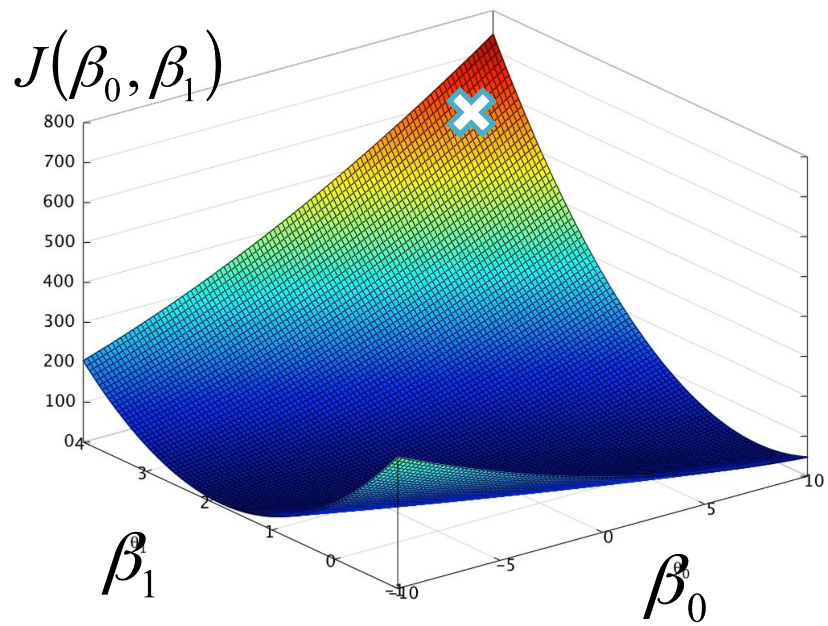
How can we do this?
(without seeing the graph of J(β)!)

Start with the function J(β):

$$J(\beta_0, \beta_1) = 1/2 \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$

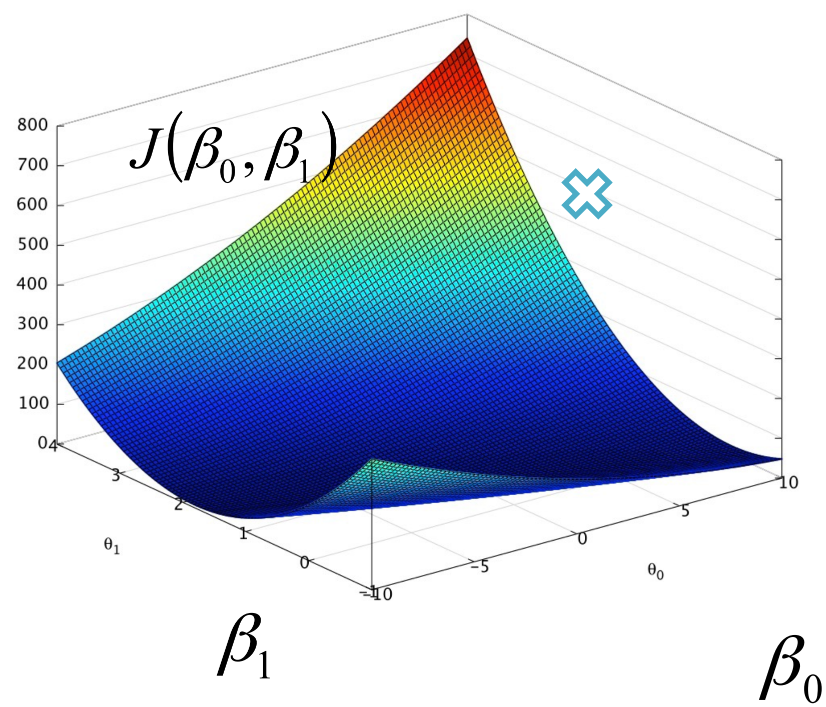
and compute its
<u>gradient vector</u> $\nabla J(\beta)$.

The gradient points in the "direction of maximum increase" of J.



Gradient Descent with Linear Regression

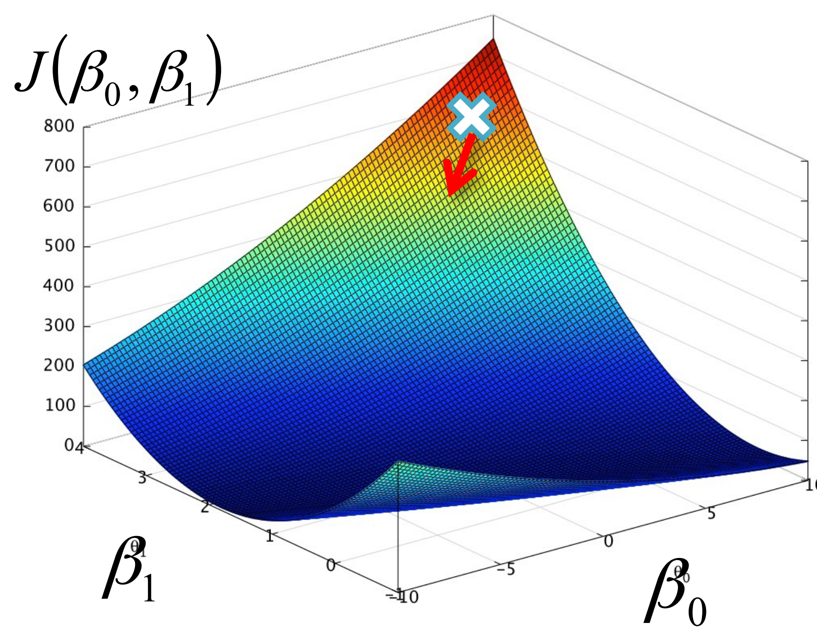
$$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$



Gradient Descent with Linear Regression

$$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

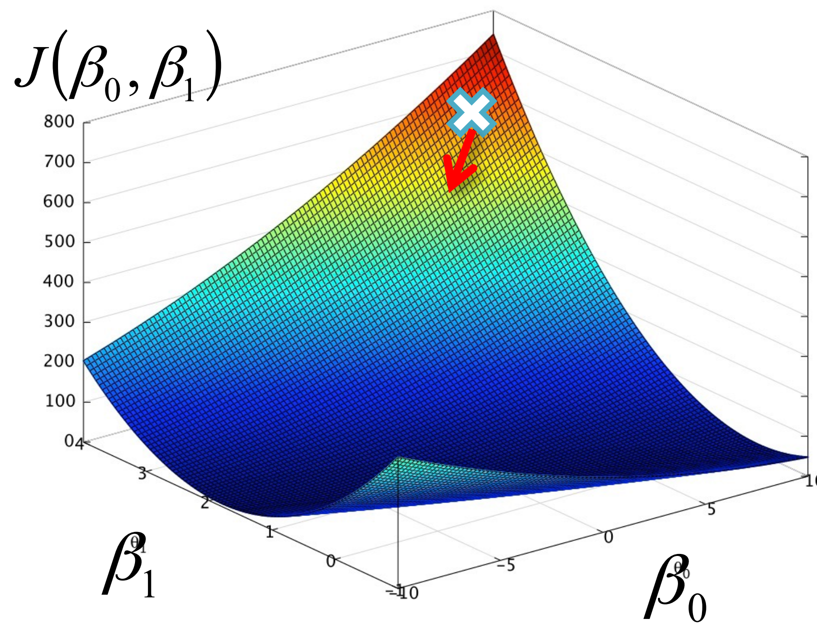
$$w_1=w_0-\alpha\nabla 1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$



Gradient Descent with Linear Regression

$$J\left(\beta_0, \beta_1\right) = 1/2\sum_{i=1}^{m}\left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2$$

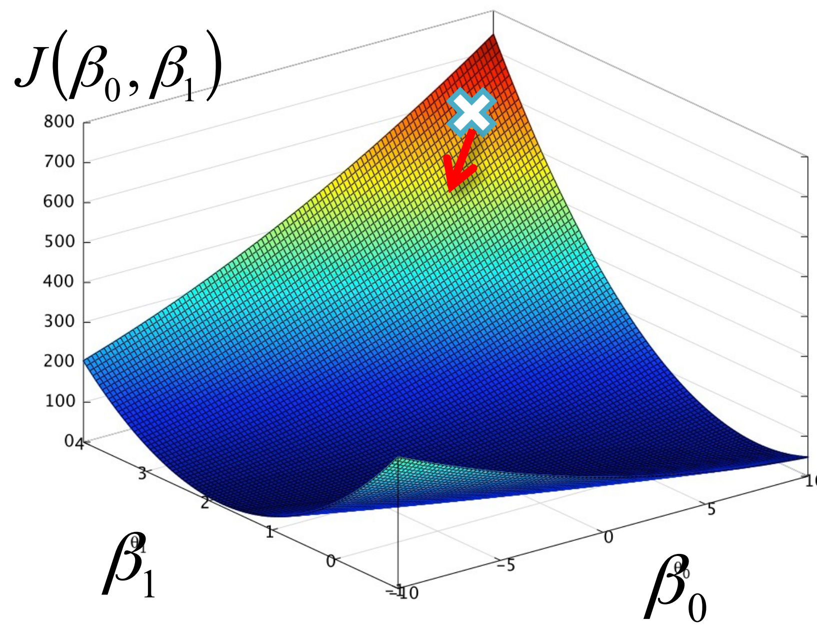
$$w_1 = w_0 - \alpha\left(\frac{\partial}{\partial\beta_0}, \cdots, \frac{\partial}{\partial\beta_n}\right)1/2\sum_{i=1}^{m}\left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2$$



Gradient Descent with Linear Regression

$$J\left(\beta_0, \beta_1\right) = 1/2 \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$

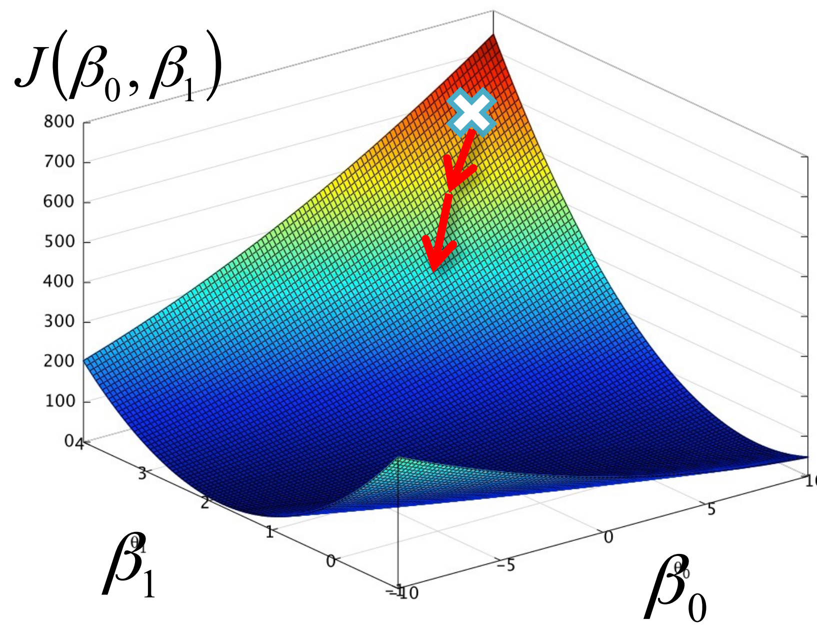
$$w_1 = w_0 - \alpha \nabla 1/2 \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$



Gradient Descent with Linear Regression

$$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

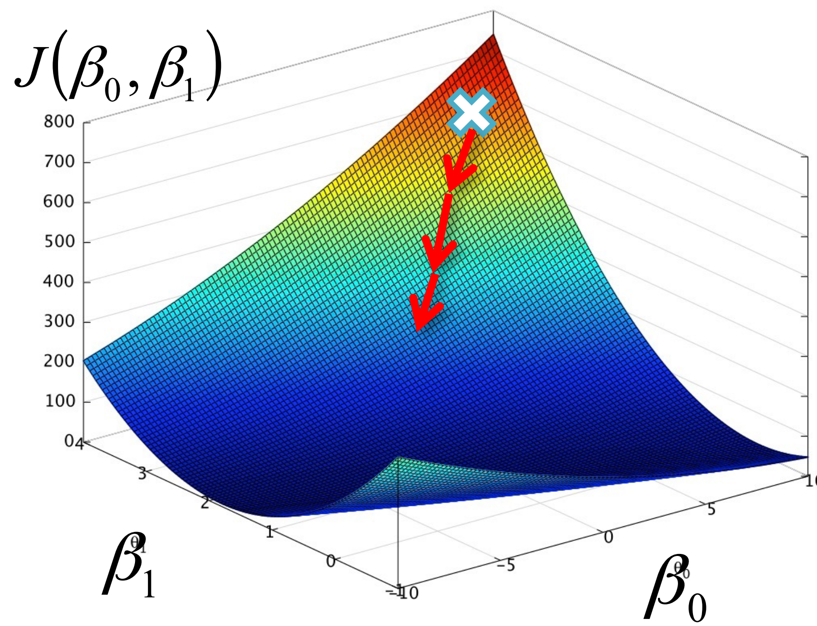
$$w_2=w_1-\alpha\nabla 1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$



Gradient Descent with Linear Regression

$$J\left(\beta_0, \beta_1\right) = 1/2\sum_{i=1}^{m}\left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2$$

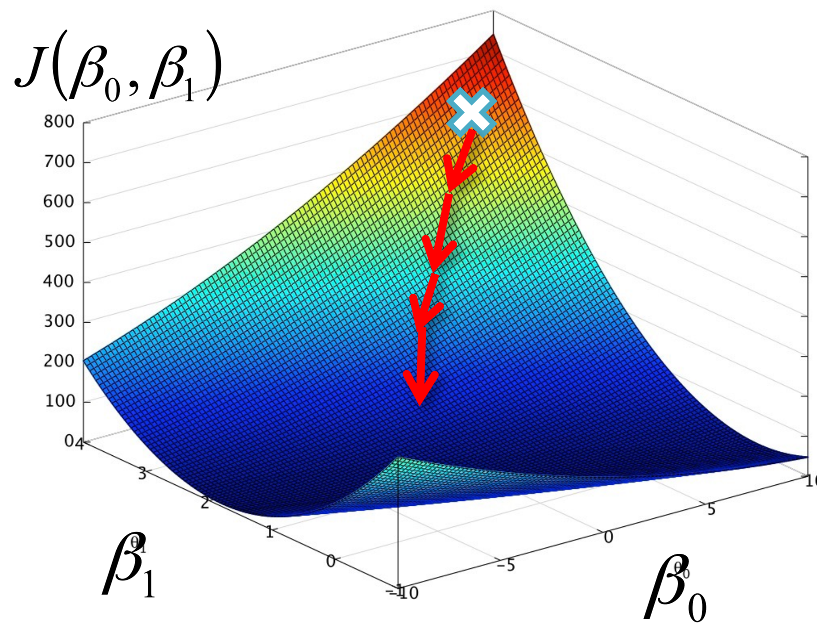
$$w_3 = w_2 - \alpha\nabla 1/2\sum_{i=1}^{m}\left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2$$



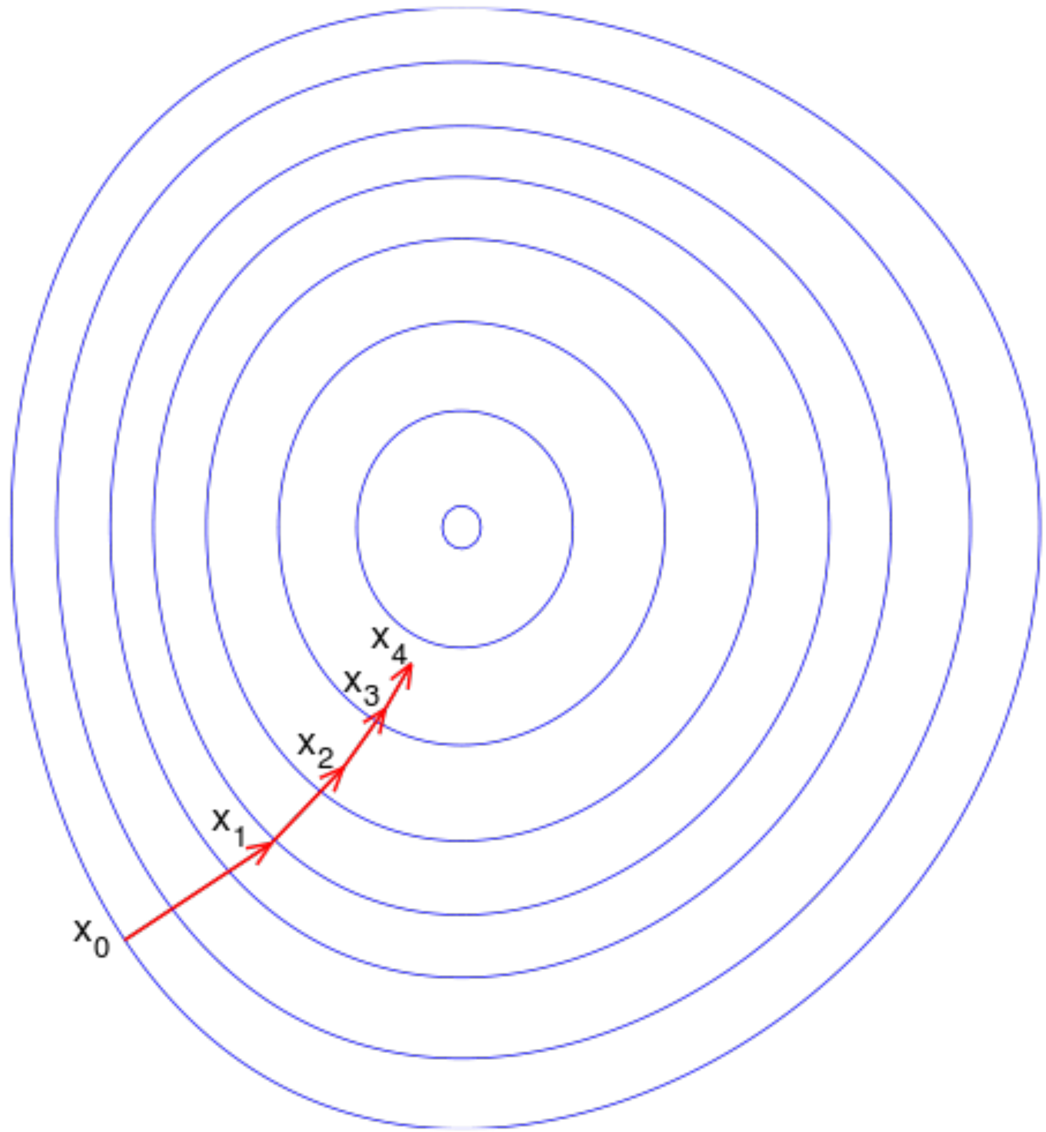
Gradient Descent with Linear Regression

$$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

$$w_4 = w_3 - \alpha\nabla 1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$



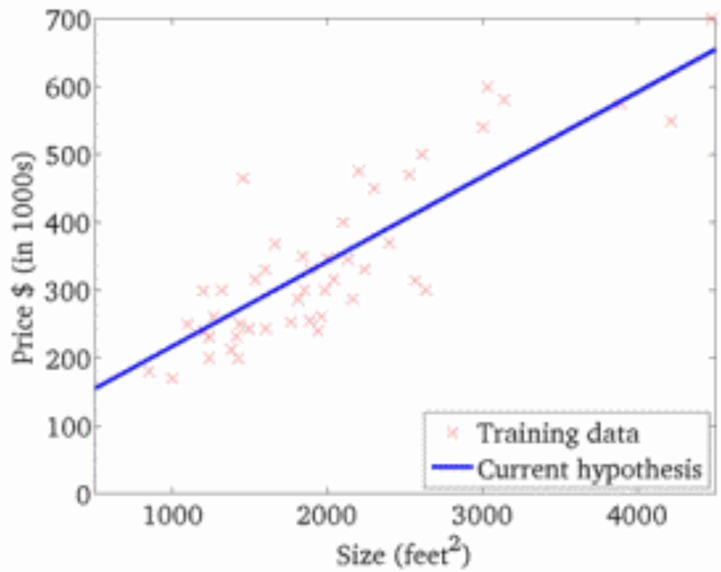
Gradient Descent



Gradient Descent

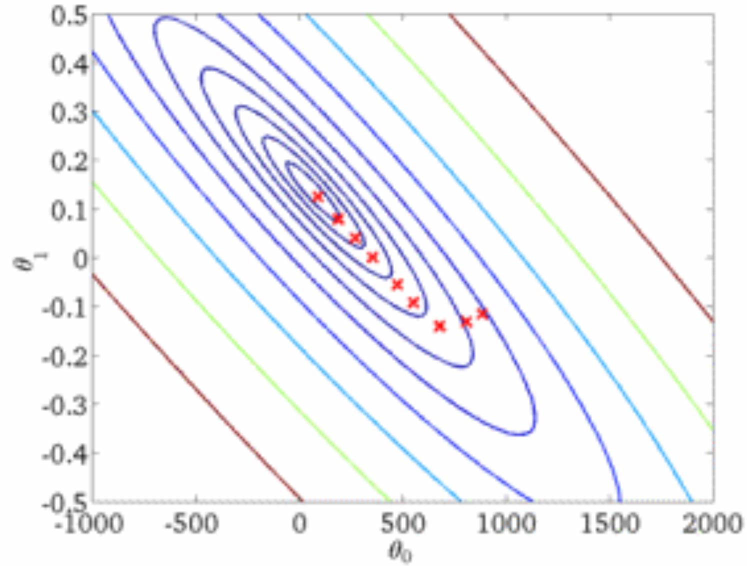
$h_\theta(x)$

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



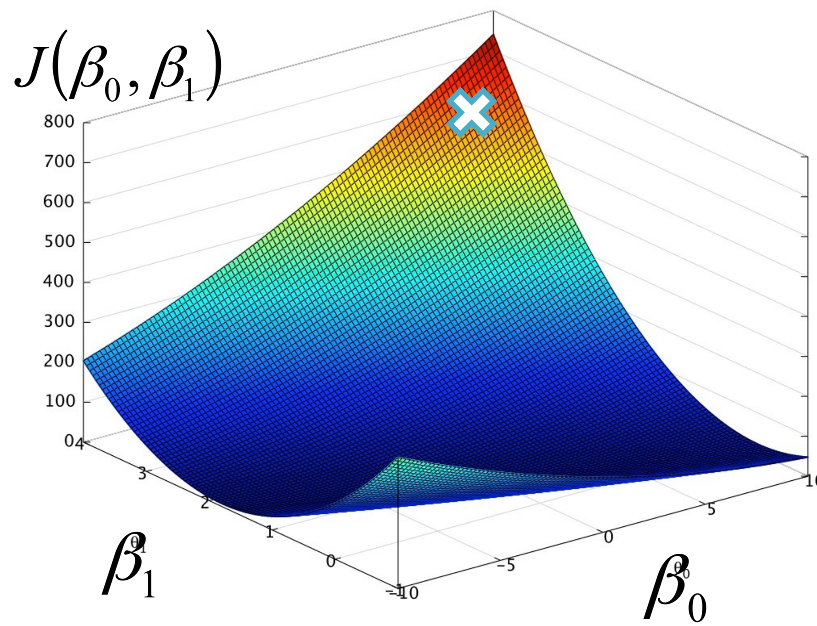
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Stochastic Gradient Descent

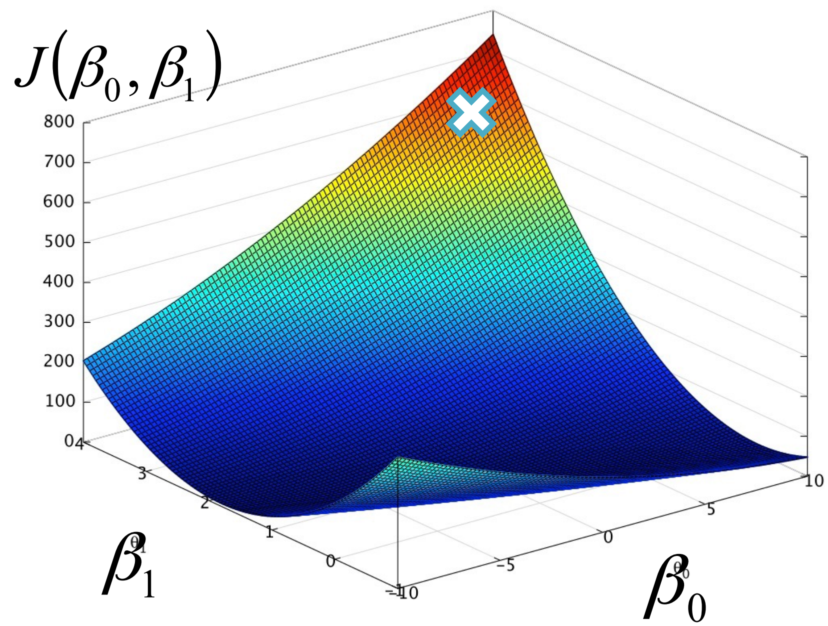
$$J\left(\beta_0, \beta_1\right) = 1/2 \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$



Stochastic Gradient Descent

$$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

Instead of using all points to find the gradient,
Only use a SINGLE point each time

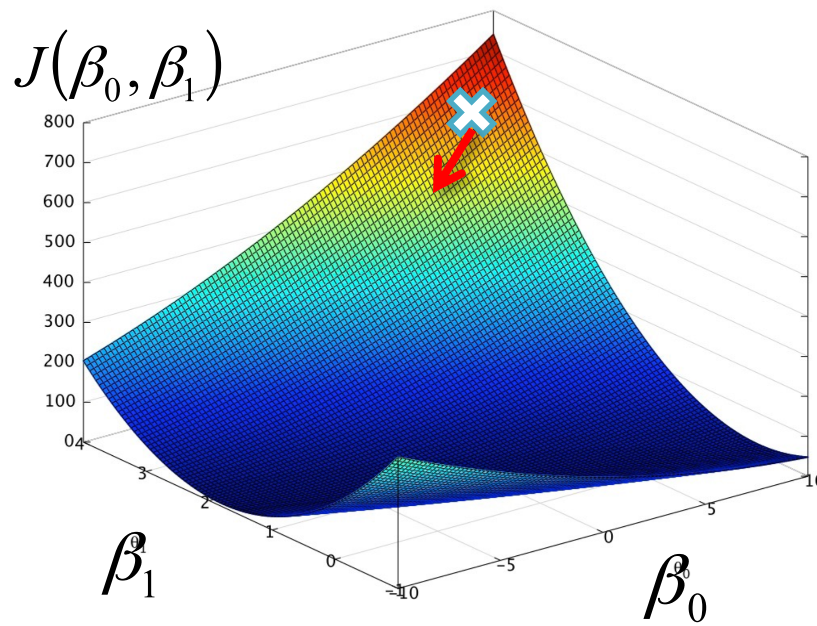


Stochastic Gradient Descent

$$J(\beta_0, \beta_1) = 1/2 \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$

Instead of using all points to find the gradient,
Only use a SINGLE point each time

$$w_1 = w_0 - \alpha \nabla 1/2 \left((\beta_0 + \beta_1 x_{obs}^{(0)}) - y_{obs}^{(0)} \right)^2$$

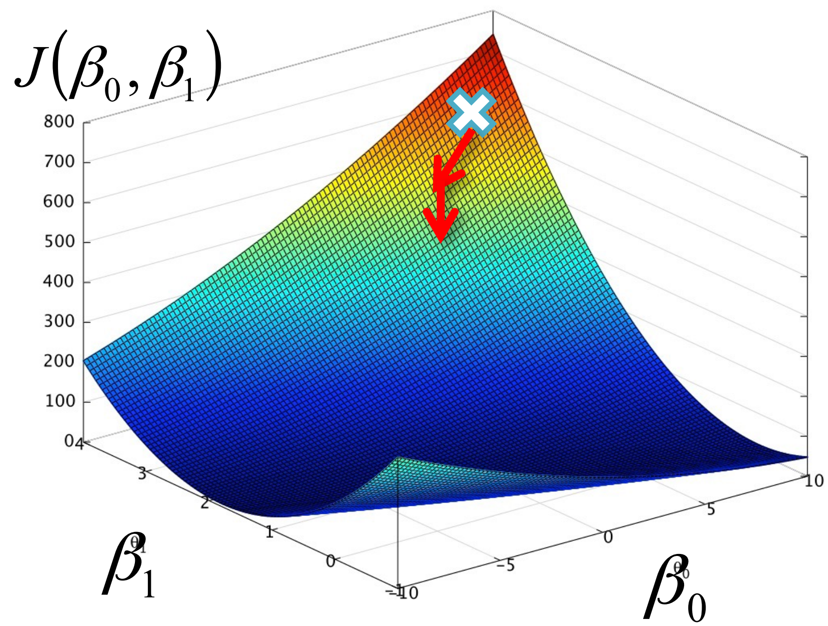


Stochastic Gradient Descent

$$J(\beta_0, \beta_1) = 1/2 \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$

Instead of using all points to find the gradient,
Only use a SINGLE point each time

$$w_2 = w_1 - \alpha \nabla 1/2 \left((\beta_0 + \beta_1 x_{obs}^{(1)}) - y_{obs}^{(1)} \right)^2$$

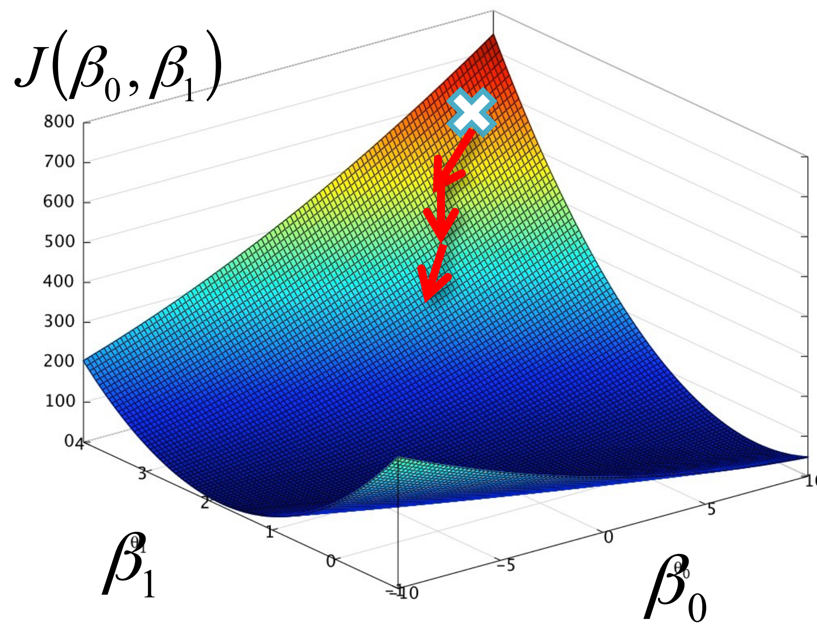


Stochastic Gradient Descent

$$J\left(\beta_0,\beta_1\right) = 1/2\sum_{i=1}^{m}\left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2$$

Instead of using all points to find the gradient,
Only use a SINGLE point each time

$$w_3 = w_2 - \alpha\nabla 1/2\left((\beta_0 + \beta_1 x_{obs}^{(2)}) - y_{obs}^{(2)}\right)^2$$

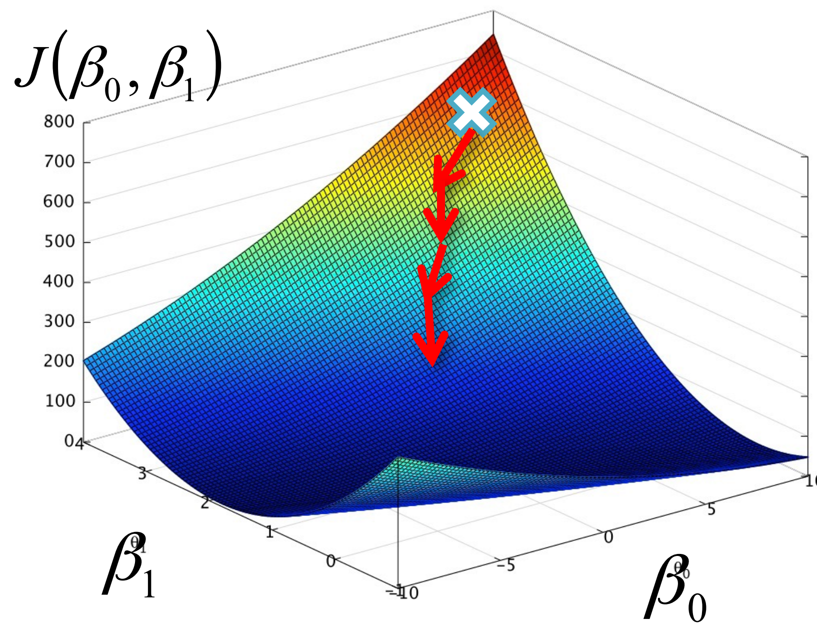


Stochastic Gradient Descent

$$J(\beta_0,\beta_1)=\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

Instead of using all points to find the gradient,
Only use a SINGLE point each time

$$w_4 = w_3 - \alpha\nabla 1/2\left((\beta_0+\beta_1 x_{obs}^{(3)})-y_{obs}^{(3)}\right)^2$$



Faster

Derivative of single point at each step (instead of 100K)

Online Training

Only need to keep single point in memory

No need to store 100K rows, large data no problem

Covers Many Algorithms

Gradient Descent is the bottleneck for linear algorithms

Can do Linear Regression, Logistic Regression, SVMs

Some Implementations

Some Implementations

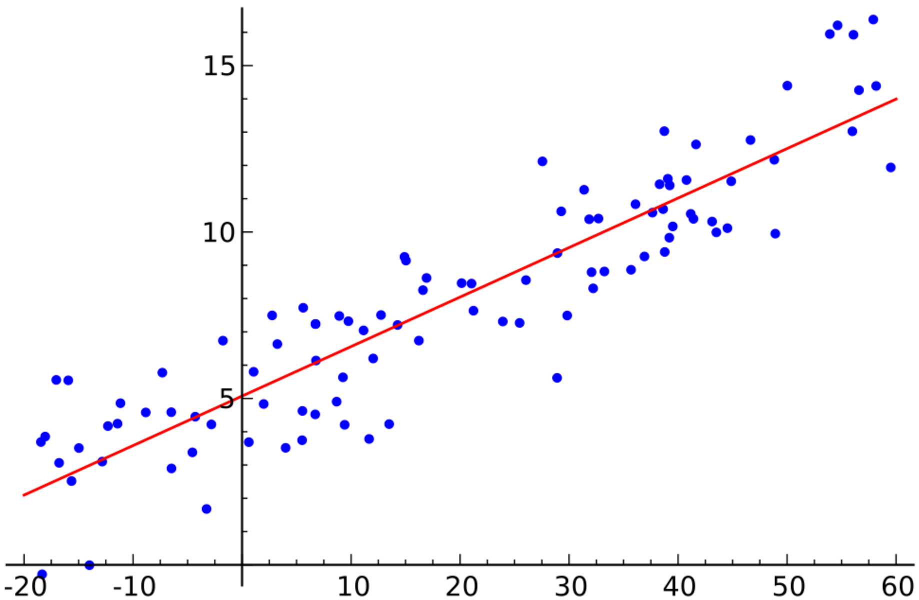
```
from sklearn.linear_model import SGDRegressor

from sklearn.linear_model import SGDClassifier
```

```python
from sklearn.linear_model import SGDRegressor
```

from sklearn.linear_model **import** SGDRegressor

SGDRegressor(loss='squared_loss')



Sum of squared errors

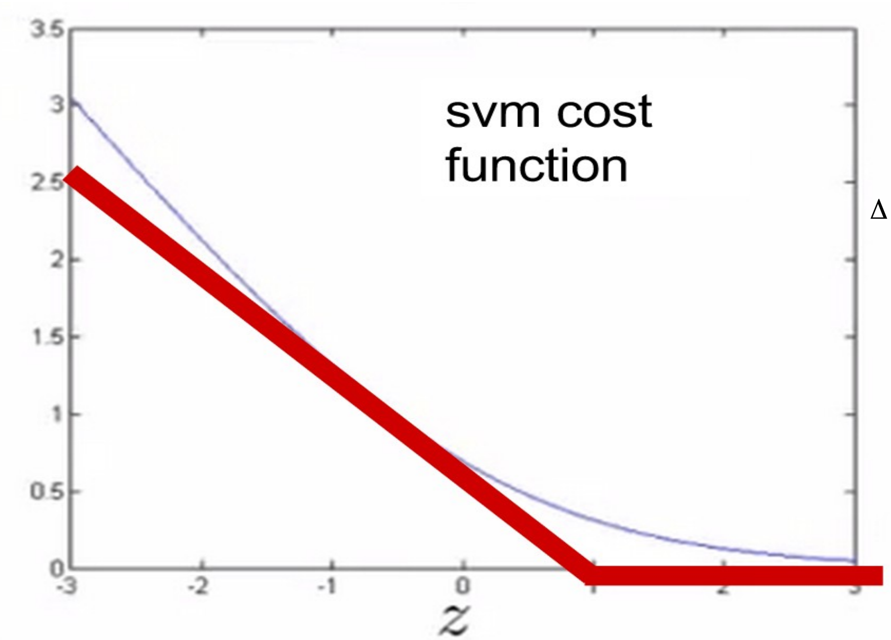
squared loss ==
Linear Regression

Optimization Function:

$$J(\beta_0, \beta_1) = 1/2 \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$

```
from sklearn.linear_model import SGDClassifier
```

```
SGDClassifier(loss='hinge')
```



Δ

Looks like a hinge.

hinge loss == SVM

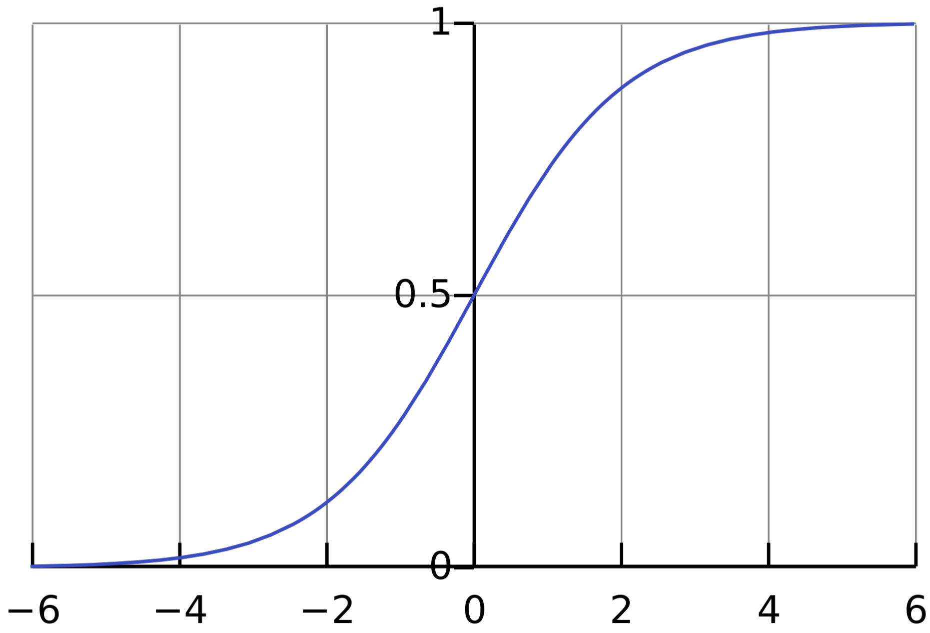
Optimization Function:

Li= max(0, Δ -yHat*y)

What is Δ ? Just like with λ within our regularization term, Δ it affects the trade-off between our data loss and our regularization loss within our objective function.

from sklearn.linear_model **import** SGDClassifier

SGDClassifier(loss='log')



This one's kind of clear

log loss == Logistic Regression

Optimization Function:

$$Li = -\log\left(\frac{e^{f_{yi}}}{\sum_j e^{f_j}}\right)$$

where

$$e^{f_{yi}} = \exp(\beta_0 + \omega^T x_{obs}^{(i)})$$

```
from sklearn.linear_model import SGDClassifier

SGDClassifier(alpha=0.0001,
              penalty='l2',
              l1_ratio=0.15)
```

Regularization parameters

Penalty values: 'l1', 'l2', 'elasticnet'

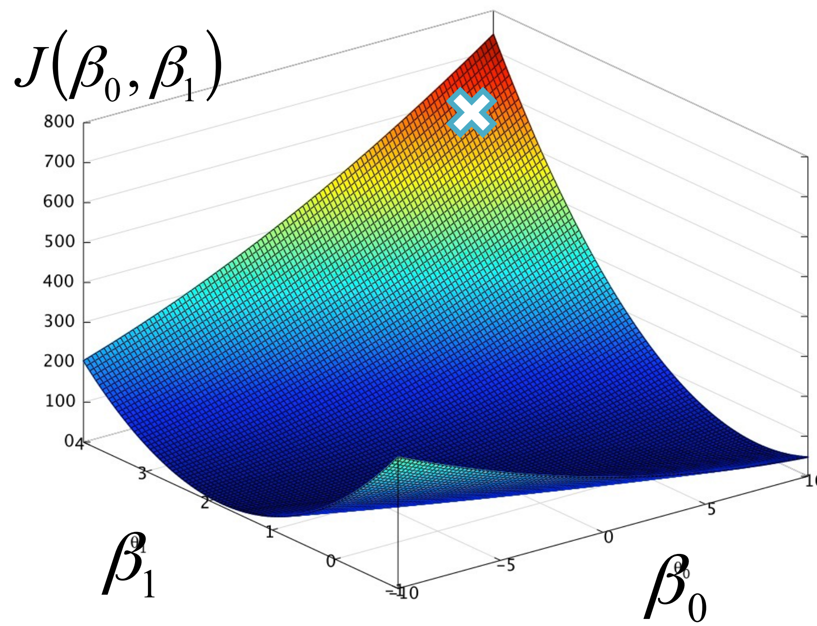
L1 optimiz. Function: $$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2-\alpha\sum_{j=1}^{k}|\beta_j|$$

L2 optimiz. Function: $$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2-\alpha\sum_{j=1}^{k}\beta_j^2$$

Mini Batch Gradient Descent

$$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

Combines of the best of both worlds ('Vanilla' Gradient Descent & Stochastic Gradient Descent): performs an update for every n training examples.

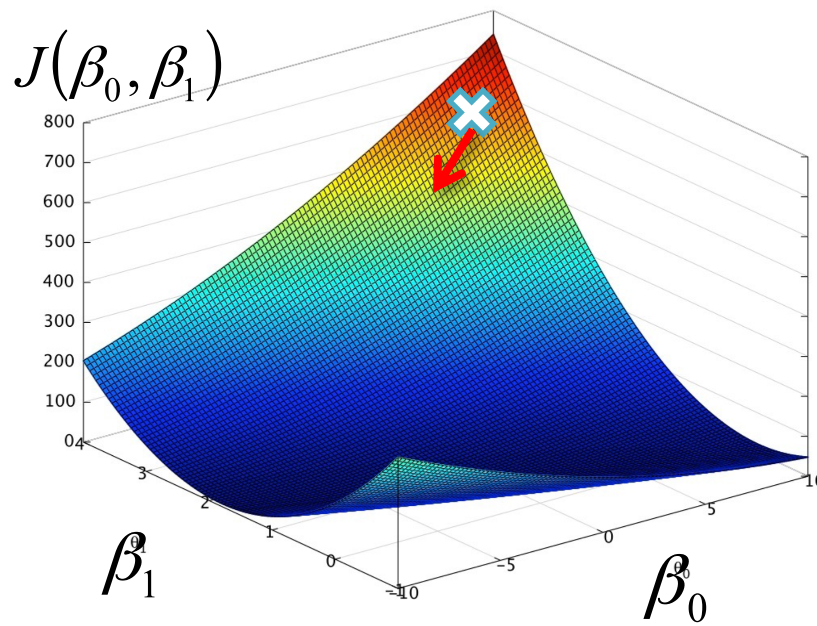


Mini Batch Gradient Descent

$$J\left(\beta_0,\beta_1\right)=1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

Combines of the best of both worlds ('Vanilla' Gradient Descent & Stochastic Gradient Descent): performs an update for every n training examples.

$$w_1=w_0-\alpha\nabla 1/2\sum_{i=1}^{n}\left((\beta_0+\beta_1 x_{obs}^{(i)})-y_{obs}^{(i)}\right)^2$$

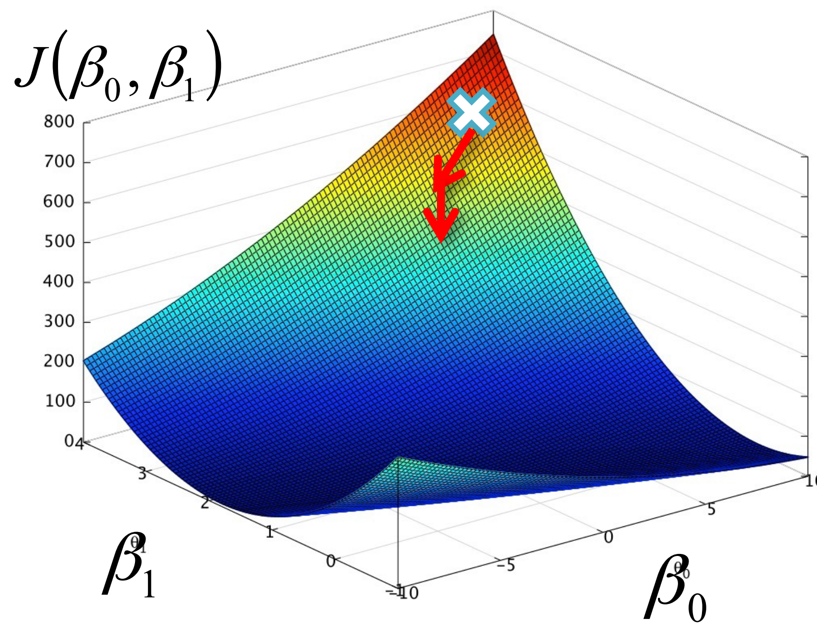


Mini Batch Gradient Descent

$$J\left(\beta_0,\beta_1\right) = 1/2\sum_{i=1}^{m}\left((\beta_0+\beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2$$

Combines of the best of both worlds ('Vanilla' Gradient Descent & Stochastic Gradient Descent): performs an update for every n training examples.

$$w_2 = w_1 - \alpha\nabla 1/2\sum_{i=1}^{n}\left((\beta_0+\beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2$$

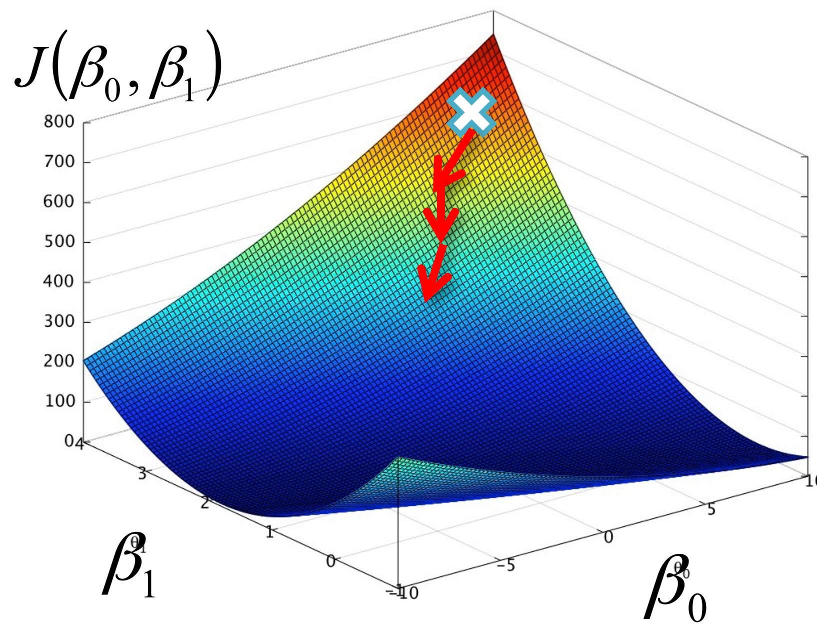


Mini Batch Gradient Descent

$$J\left(\beta_0, \beta_1\right) = 1/2 \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$

Combines of the best of both worlds ('Vanilla' Gradient Descent & Stochastic Gradient Descent): performs an update for every n training examples.

$$w_3 = w_2 - \alpha \nabla 1/2 \sum_{i=1}^{n} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)} \right)^2$$



Mini Batch Gradient Descent

- Mini Batch implementation is typically used for neural nets
- Mini Batch sizes typically range from 50 to 256.
- There is a trade off between MB size and the learning rate (α): we can reduce learning rate for larger mini batch sizes (vice versa)
- We can tailor a learning rate schedule (i.e.): we can reduce the learning rate as go further along within our training epoch.
- We can implement with SGDClassifier (using 'partial fit') ex: SGDClassifier(loss='log').partial_fit()