Gradient Descent Algorithm

Basic PPT with Fundamental Equations

Gradient Descent is often used as black-box tools

- Gradient descent is popular algorithm to perform optimization of deep learning.
  - Many Deep Learning library contains various gradient descent algorithms.
    - Example : Keras, Chainer, Tensorflow…

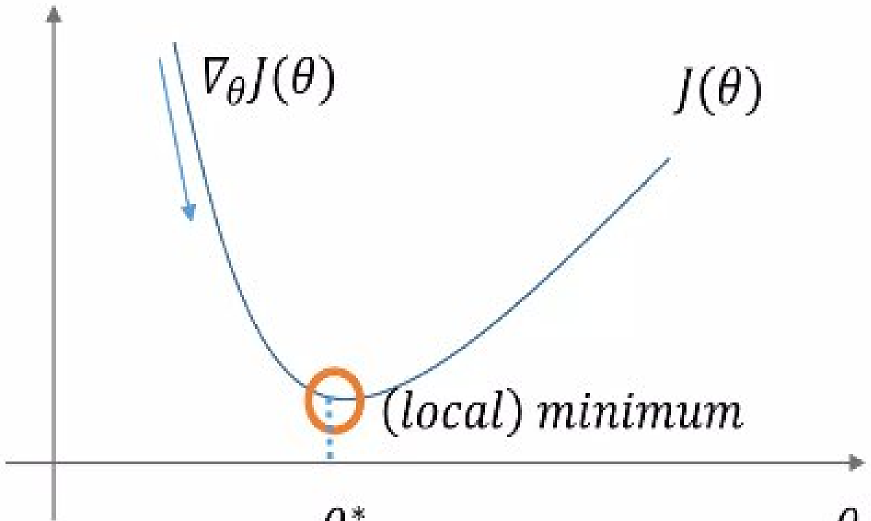
- However, **these algorithms often used as black-box tools and many people don't understand their strength and weakness**.
  - We will learn this.

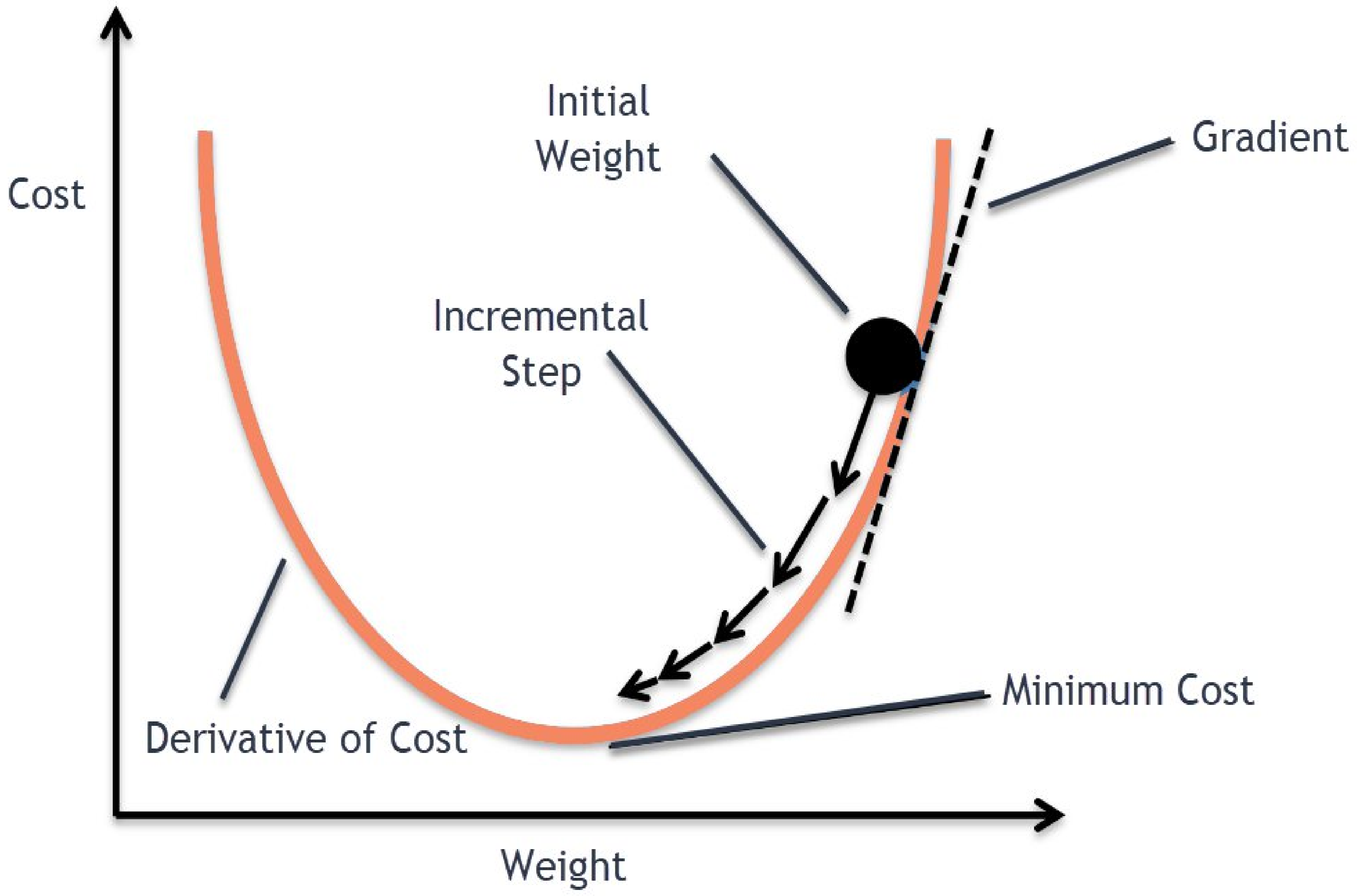
Gradient Descent

- Gradient descent is a way to minimize an objective function $J(\theta)$
  - $J(\theta)$: Objective function
  - $\theta \in R^d$: Model's parameters
  - η : Learning rate. This determines the size of the steps we take to reach a (local) minimum.

Update equation

$$\theta = \theta - \eta * \nabla_\theta J(\theta)$$




Initial Weight
Gradient
Cost
Incremental Step
Minimum Cost
Derivative of Cost
Weight

Trade-off

- Depending on the amount of data, they make a trade-off :
  - The **accuracy** of the parameter update
  - The **time** it takes to perform an update.

Method	Accuracy	Time	Memory Usage	Online Learning
Batch gradient descent	○	Slow	High	×
Stochastic gradient descent	△	High	Low	○
Mini-batch gradient descent	○	Midium	Midium	○

Batch gradient descent

This method computes the gradient of the cost function with **the entire training dataset.**

Update equation

$$\theta = \theta - \eta * \nabla_\theta J(\theta)$$

We need to calculate the gradients for the whole dataset to perform **just one update.**

Code

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

Batch gradient descent

- Advantage
  - It is guaranteed to converge **to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.**

- Disadvantages
  - It can be **very slow**.
  - It is intractable for datasets that **do not fit in memory**.
  - It **does not allow** us to update our model **online.**

Stochastic gradient descent

This method performs a parameter update for **each** training example $x^{(i)}$ and label $y^{(i)}$.

Update equation

$$\theta = \theta - \eta * \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

We need to calculate the gradients for the whole dataset to perform **just one update.**

Code

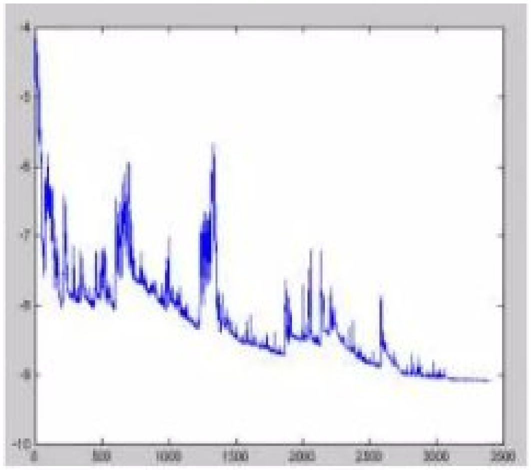
```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

Note : we shuffle the training data at every epoch

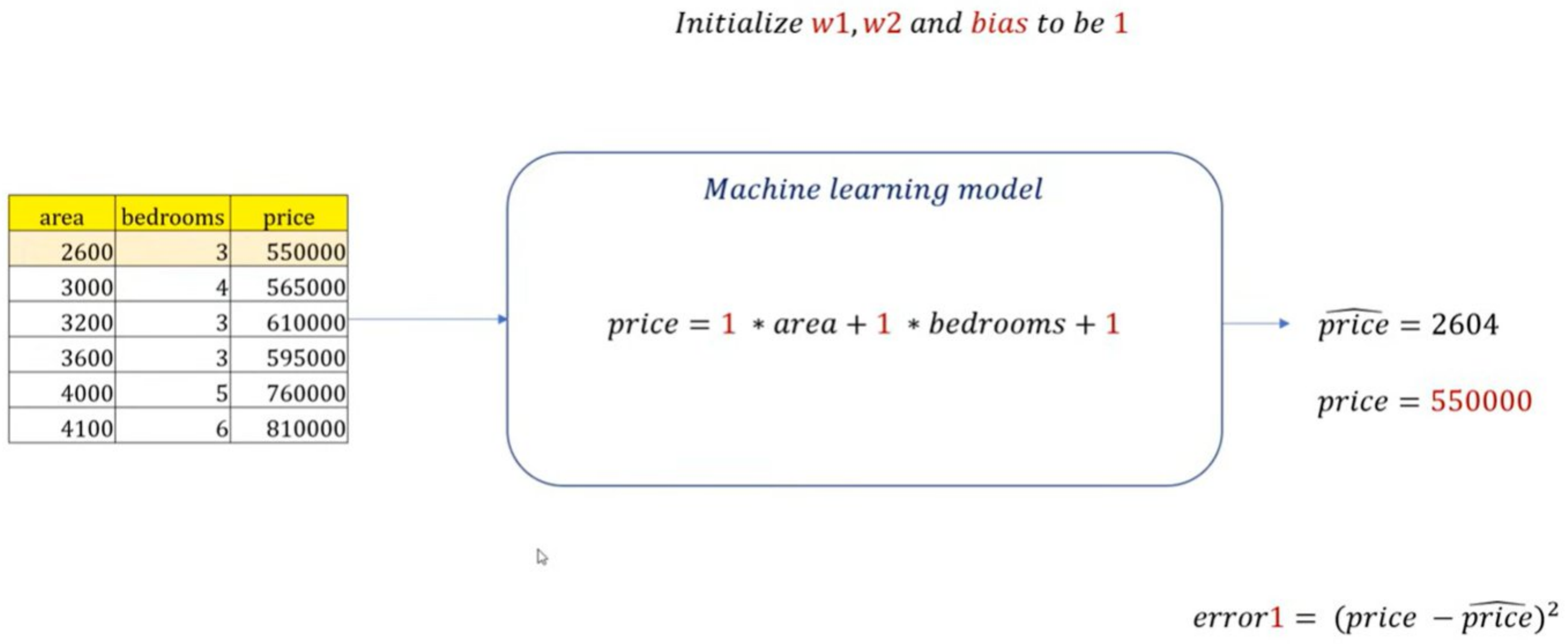
Stochastic gradient descent

- Advantage
  - It is usually **much faster** than batch gradient descent.
  - It can be **used to learn online.**

- Disadvantages
  - It performs frequent updates with a **high variance** that cause the objective function to fluctuate heavily.



Example



$Initialize\ w1, w2\ and\ bias\ to\ be\ 1$

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

Machine learning model

$$price = 1 * area + 1 * bedrooms + 1$$

$$\widehat{price} = 2604$$

$$price = 550000$$

$$error1 = (price - \widehat{price})^2$$

https://www.youtube.com/watch?v=IU5fuoYBTAM

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

Machine learning model

$$price = 1 * area + 1 * bedrooms + 1$$

$$\widehat{price} = 4107$$

$$price = 810000$$

$$error6 = (price - \widehat{price})^2$$

End of first epoch

$$Total\ Error = error1 + error2 + \ ... + error6$$

$$Mean\ Squred\ Error(a.k.a.MSE) = \frac{Total\ Error}{6}$$

$\mathrm{w1} = \mathrm{w1} - learning\ rate * {}^{\partial(\mathrm{MSE})}/_{\partial w1}$ $\mathrm{w1} = 1 - (-50) = 51$

$\mathrm{w2} = \mathrm{w2} - learning\ rate * {}^{\partial(\mathrm{MSE})}/_{\partial w2}$ $\mathrm{w2} = 1 - (-8) = 9$

$\mathrm{b} = \mathrm{b} - learning\ rate * {}^{\partial(\mathrm{MSE})}/_{\partial b}$ $\mathrm{bias} = 1 - (-20000) = 20001$

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

Machine learning model

$$price = 51 * area + 9 * bedrooms + 20001$$

$$\widehat{price} = 229155$$

$$price = 550000$$

$$error6 = (price - \widehat{price})^2$$

End of second epoch

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

$10\ million\ samples$

1) To find cumulative error for first round (epoch) now we need to do a forward pass for 10 million samples
2) We have 2 features (area and bedroom). This requires finding 20 million derivatives

1. *Randomly pick single data training sample*

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

10 *million samples*

Machine learning model

$$price = 1 * area + 1 * bedrooms + 1$$

$$\widehat{price} = 3204$$

$$price = 610000$$

$$error = (price - \widehat{price})^2$$

$2. \textit{Adjust weights}$

$$w1 = w1 - \textit{learning rate} * {}^{\partial(\text{error})}/_{\partial w1} \qquad w1 = 1 - (-13) = 14$$

$$w2 = w2 - \textit{learning rate} * {}^{\partial(\text{error})}/_{\partial w2} \qquad w2 = 1 - (-3) = 4$$

$$b = b - \textit{learning rate} * {}^{\partial(\text{error})}/_{\partial b} \qquad \text{bias} = 1 - (-1500) = 1501$$

3. *Again randomly pick a training sample*

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

10 million samples

Machine learning model

$$price = 14 * area + 4 * bedrooms + 1501$$

$$\widehat{price} = 3204$$

$$price = 37913$$

$$error = (price - \widehat{price})^2$$

$4. \, Again \; adjust \; weights$

$$\mathrm{w1} = \mathrm{w1} - learning\,rate * {}^{\partial(\mathrm{error})}/_{\partial w1}$$

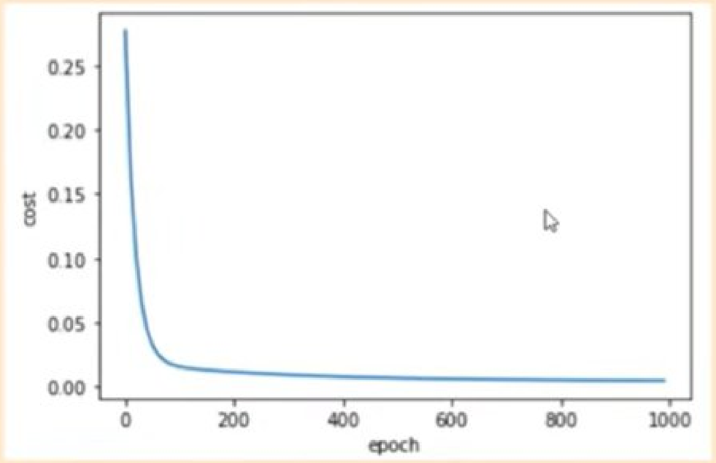
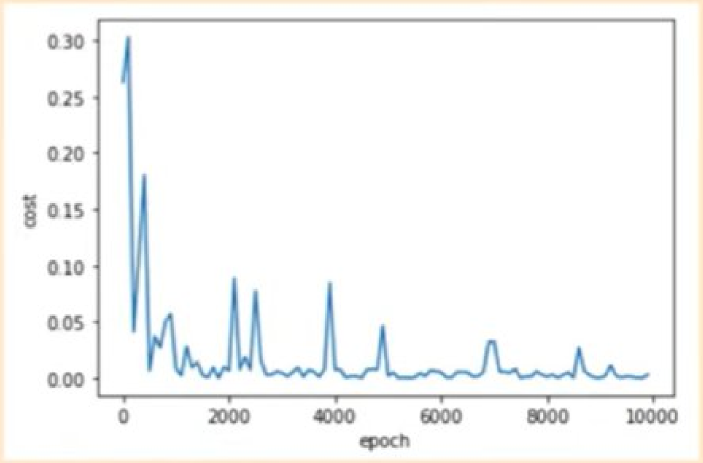
$$\mathrm{w1} = 14 - (-100) = 114$$

$$\mathrm{w2} = \mathrm{w2} - learning\,rate * {}^{\partial(\mathrm{error})}/_{\partial w2}$$

$$\mathrm{w2} = 4 - (-12) = 16$$

$$\mathrm{b} = \mathrm{b} - learning\,rate * {}^{\partial(\mathrm{error})}/_{\partial b}$$

$$\mathrm{bias} = 1501 - (-2001) = 3502$$

Batch Gradient Descent	Stochastic Gradient Descent (SGD)
Use **all** training samples for one forward pass and then adjust weights	Use **one** (randomly picked) sample for a forward pass and then adjust weights
Good for small training set	Good when training set is very big and we don't want too much computation
	

Mini batch is like SGD. Instead of choosing one randomly picked training sample, you will use a batch of randomly picked training samples.

1. For example I have 20 training samples total.
2. Let's say I use 5 random samples for one forward pass to calculate cumulative error
3. After that I adjust weights

Gradient Descent Comparison Table (Updated)

Feature	Batch Gradient Descent (BGD)	Stochastic Gradient Descent (SGD)	Mini-Batch Gradient Descent (MBGD)
Gradient Computed On	Entire dataset	One sample at a time	A small subset (batch) of data (e.g., 32, 64, 128)
Updates per Epoch	1 update	N updates (N = no. of samples)	N / batch_size updates
Speed	Slow	Fast	Faster than BGD, slower than SGD
Memory Requirement	High	Very low	Moderate
Convergence	Smooth & stable	Noisy, oscillates	Balanced, smooth but faster
Accuracy	High (best gradient estimate)	Lower due to noise	High (close to BGD)
Suitable For	Small–medium datasets	Very large datasets, online learning	Most practical ML problems
Pros	Stable convergence, full-data accuracy	Very fast updates, low memory	Good trade-off between speed & accuracy
Cons	Computationally expensive	Noisy updates, may overshoot	Requires tuning batch size