Gradient Descent

CS530
Chapman

Spring 2020

1

GRADIENT DESCENT

2

Hiker in the fog

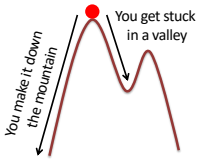
Imagine you are hiking in the fog and need to get down a mountain.

You cannot see where the optimal downhill path it. But you can see your immediate surroundings. So you decide to choose the steepest downhill path at any point.

If you are lucky, this method will get you down the hill. If not, you might get stuck in some valley or ravine, still up the mountain.

Taking the best possible downhill path at any point is the essence of the gradient descent algorithm.

Image source



3

Gradient descent (GD)

Also called batch gradient-descent.

For (smooth) function f of x, and for a small ε:

$$f(x+\varepsilon) \approx f(x) + \varepsilon f'(x)$$

So, the derivative specifies how to scale small changes in input, from x to $x+\varepsilon$, to corresponding change in output, from $f(x)$ to $f(x+\varepsilon)$.

Hence, moving in small steps opposite to the derivative sign would reduce f.

4

GD example

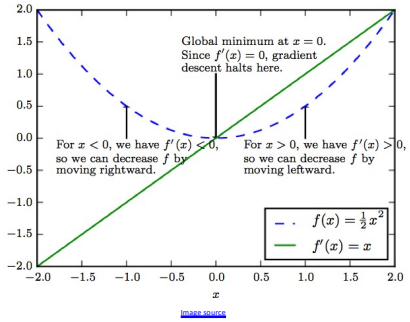


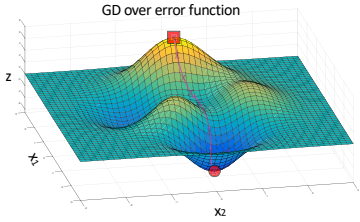
Image source

5

GD for multidimensional functions

GD works for mutli-dimensional functions too

$$z = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$



6

GD for multidimensional functions

For multivariate functions $f(\vec{x}) = f(x_1, x_2, \cdots, x_n)$, we must make use of partial derivatives $\frac{\partial}{\partial x_i} f(\vec{x})$ to find gradients.

Partial derivatives measure how f changes when only variable x_i changes.

Gradient generalizes the notion of derivative when differentiating over vectors.

Gradient of f is vector, denoted $\nabla_{\vec{x}} f(\vec{x})$, containing all partial derivatives of f with respect to $\vec{x}$. So

$$\nabla_{\vec{x}} f(\vec{x}) = \left(\frac{\partial}{\partial x_1} f(\vec{x}), \frac{\partial}{\partial x_2} f(\vec{x}), \cdots, \frac{\partial}{\partial x_n} f(\vec{x}) \right).$$

7

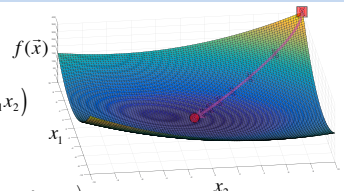
GD for multidimensional functions

Example:

For

$$f(\vec{x}) = f\left(x_1^2 + 2x_2^2 + x_1 x_2\right)$$

then



$$\nabla_{\vec{x}} f(\vec{x}) = \left(\frac{\partial}{\partial x_1} f(\vec{x}), \frac{\partial}{\partial x_2} f(\vec{x}), \cdots, \frac{\partial}{\partial x_n} f(\vec{x}) \right)$$

$$\nabla_{\vec{x}} f(\vec{x}) = \left(2x_1 + x_2, 4x_2 + x_1\right)$$

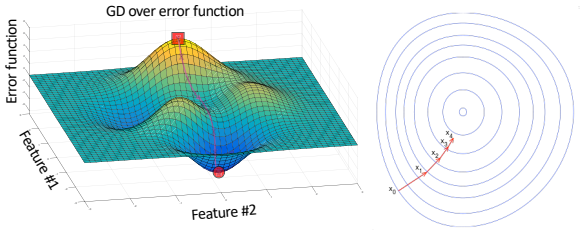
So, descending the gradient of f is: $-\nabla_{\vec{x}} f(\vec{x}) = \left(-2x_1 - x_2, -4x_2 - x_1\right)$.

And GD algorithm would move from $\vec{x}_0$ to $\vec{x}_0 - \eta \nabla_{\vec{x}} f(\vec{x}_0)$.

8

GD summary

- GD often works well, descending from "great heights of error" to "low valleys of truth"



9

STOCHASTIC GRADIENT DESCENT

10

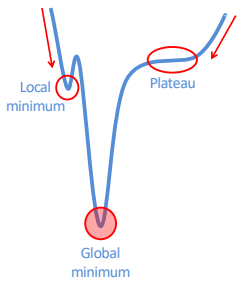
Limitations of Gradient Descent

1. GD only guaranteed to reach local minimum
   - Might reach bad local minimum
   - Might not get past plateaus
2. GD does not scale well with size of dataset
   - Prohibitively time-consuming to compute over large datasets
3. Sometimes GD zigzags slowly instead of heading straight toward minimum
4. Training-set minimum might over-fit, so not generalize well

11

1. GD & local minima

1. GD only guaranteed to reach local minimum
   - Might reach bad local minimum
   - Might not get past plateaus
2. GD does not scale well with size of dataset
   - Prohibitively time-consuming to compute over large datasets
3. Sometimes GD zigzags slowly instead of heading straight toward minimum
4. Training-set minimum might over-fit, so not generalize well



12

2. GD prohibitive for large datasets

Cost function is typically average over training all m samples of per-sample loss function.

For example, for loss function L,

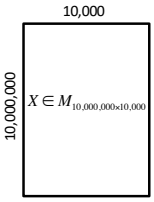
$$J(\vec{\vartheta}) = \frac{1}{m}\sum_{i=1}^{m} L(\vec{x}^{(i)}, y^{(i)}, \vec{\vartheta}),$$

where $\vec{\vartheta}$ are a set of parameters.

So, the gradient is

$$\nabla_{\vec{\vartheta}} J(\vec{\vartheta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\vec{\vartheta}} L(\vec{x}^{(i)}, y^{(i)}, \vec{\vartheta}).$$

Each gradient step thus takes $O(m)$ time, making GD unsuitable for large datasets.

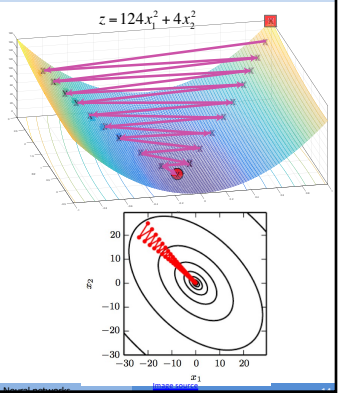


13

3. GD zigzagging toward minimum

GD always follows steepest descent, even when it leads it away from the minimum

For example: in a function with long canyon, GD might scale up and down canyon walls rather than toward the minimum.

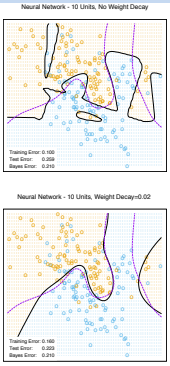


14

4. Stopping before the minimum

Often model parameters that minimize the error function over the training set do not generalize well outside the training set. In such cases. the model parameters greatly depend on the specific training set used. This is termed over-fitting, as we discussed.

We therefore often want to stop training our model before reaching the training-set minimum, either with early stopping or with weight decay.



15

What is stochastic gradient descent (SGD)?

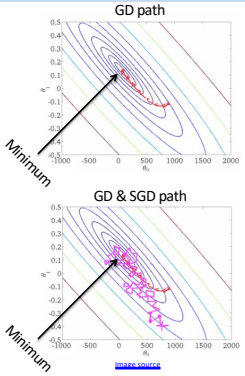
Approximate gradient using small set of samples: unbiased "minibatch", drawn uniformly, in each step. Sample size, m', is 1 to at most in the hundreds.

Approximate gradient is

$$g(\vec{\vartheta}) = \frac{1}{m'}\sum_{i=1}^{m'} \nabla_{\vec{\vartheta}} L(x^{(i)}, y^{(i)}, \vec{\vartheta}).$$

And, in the general sense, it directs GD downhill.

Importantly, we do not scale m' with dataset size.



16

Properties of SGD

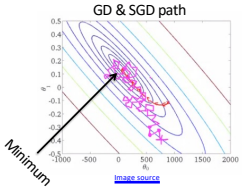
Running time of one step of SGD is $O(m') << O(m)$.

SDG generally moves in the direction of the minimum, but not always.

SGD is not guaranteed to converge to global, or even local, minimum, and typically does not converge.

However, model parameters do end up in the general vicinity that leads to minimizing the error function over the training set.

Such parameters often lead to a model that generalizes better than the parameters that lead to the global minimum over the training set.

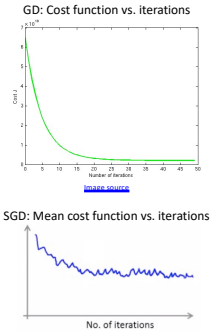


17

Checking for convergence: GD & SGD

For GD, the cost function decreases with every iteration. So we can stop when the difference in cost function between successive iterations is less than some tolerance.

SGD does not necessarily decrease the cost function in every iteration. But the cost function should decrease on average. Thus, if we check the average cost function over every n (e.g., $n = 100$) iteration steps, we expect to see a decrease.

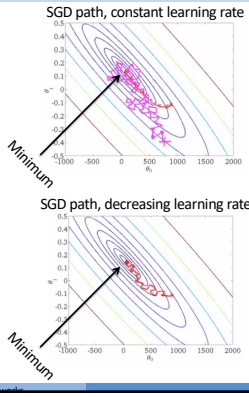


18

Learning rate in SGD

When the learning rate is constant, SGD reaches the vicinity of the cost function's minimum and then roams about.

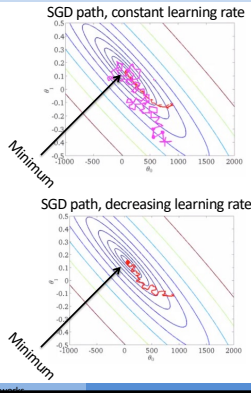
If we decrease the learning rate as a function of the iteration number, SGD will often get closer to the minimum.



19

Comparing GD & SGD

1. What is similar and what is different between the GD and SGD algorithms?
2. What are the advantages and disadvantages of GD and SGD?
3. Under which circumstances would you use each algorithm? Explain your answer.



20

GRADIENT DESCENT ON A PERCEPTRON

21

GD on NN: Widrow-Hoff (delta) rule

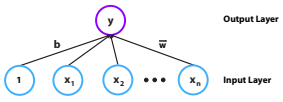
The original Perceptron had a threshold output neuron. But what about a Perceptron with a continuous output neuron?

Its formula is:

$$y = \vec{w} \cdot \vec{x}.$$

So, we have a linear (identity) activation function. We can define the difference:

$$\delta(\vec{w}) = (y - \hat{y}) = (y - \vec{w} \cdot \vec{x}).$$



Now, let us define the error:

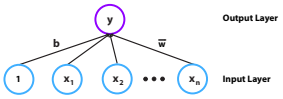
$$e(\vec{w}) = \frac{1}{2}\delta^2(\vec{w}) = \frac{1}{2}(y - \hat{y})^2$$
$$= \frac{1}{2}(y - \vec{w} \cdot \vec{x})^2.$$

22

GD on NN: Widrow-Hoff (delta) rule

We want find weights $\vec{w}$ that minimize the error, $e(\vec{w})$. So, we compute the gradient (using the chain rule):

$$\frac{\partial e(\vec{w})}{\partial \vec{w}} = \frac{\partial e(\vec{w})}{\partial \delta(\vec{w})}\frac{\partial \delta(\vec{w})}{\partial \vec{w}} = \delta(\vec{w})(-\vec{x})$$
$$= -(y - \vec{w} \cdot \vec{x})\vec{x}.$$



Now we can derive the gradient-descent learning rule:

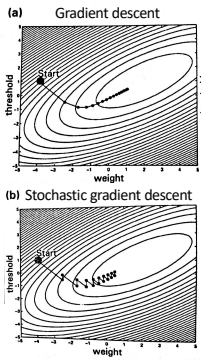
$$\Delta\vec{w} = -\eta\frac{\partial e(\vec{w})}{\partial \vec{w}} = \eta(y - \vec{w} \cdot \vec{x})\vec{x} = \eta\delta\vec{x}.$$

23

GD on NN: Widrow-Hoff (delta) rule

The delta-rule just defined operated on a single training example at a time. So, we used stochastic rather than full-batch gradient descent—an estimate of the gradient one sample at a time. The movement along this estimated gradient is more erratic, though it is roughly the same on average.

Also, the delta and perceptron update rules might appear similar. But the former is for linear neurons while the latter is for step-function neurons. The delta rule thus (1) converges faster than the perceptron rule, and (2) converges at least to a local minimum, even when the data is not linearly separable.



24