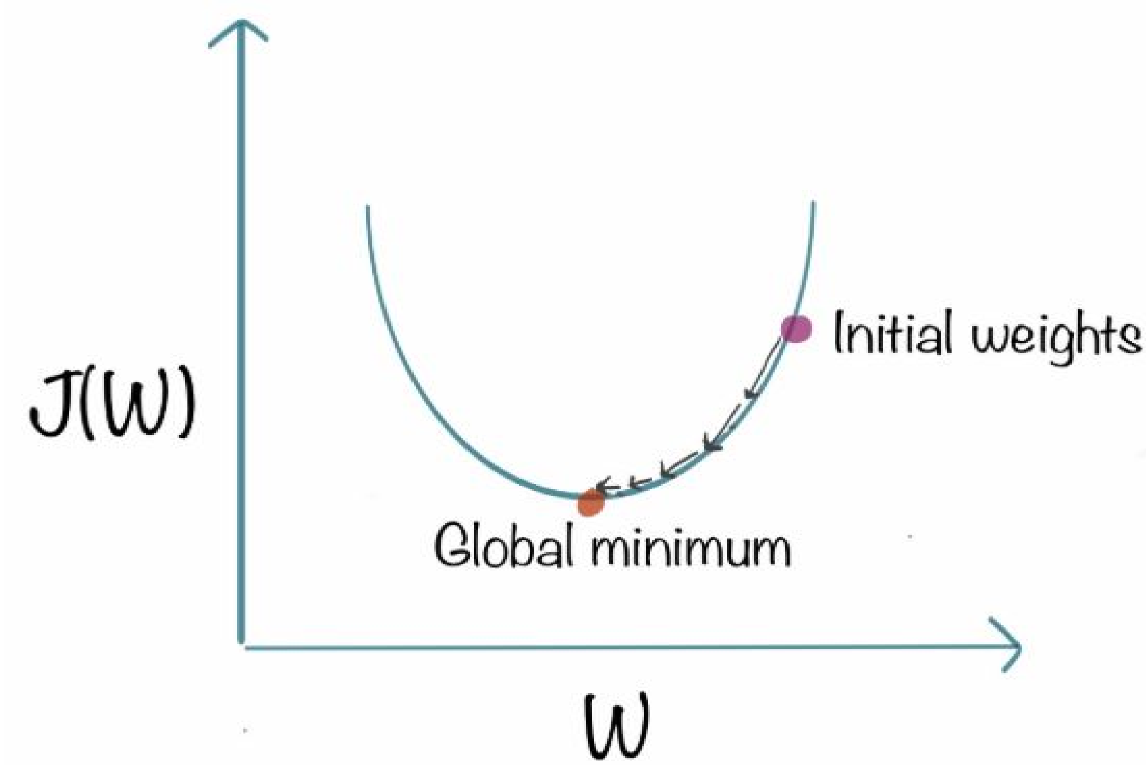
GRADIENT DESCENT

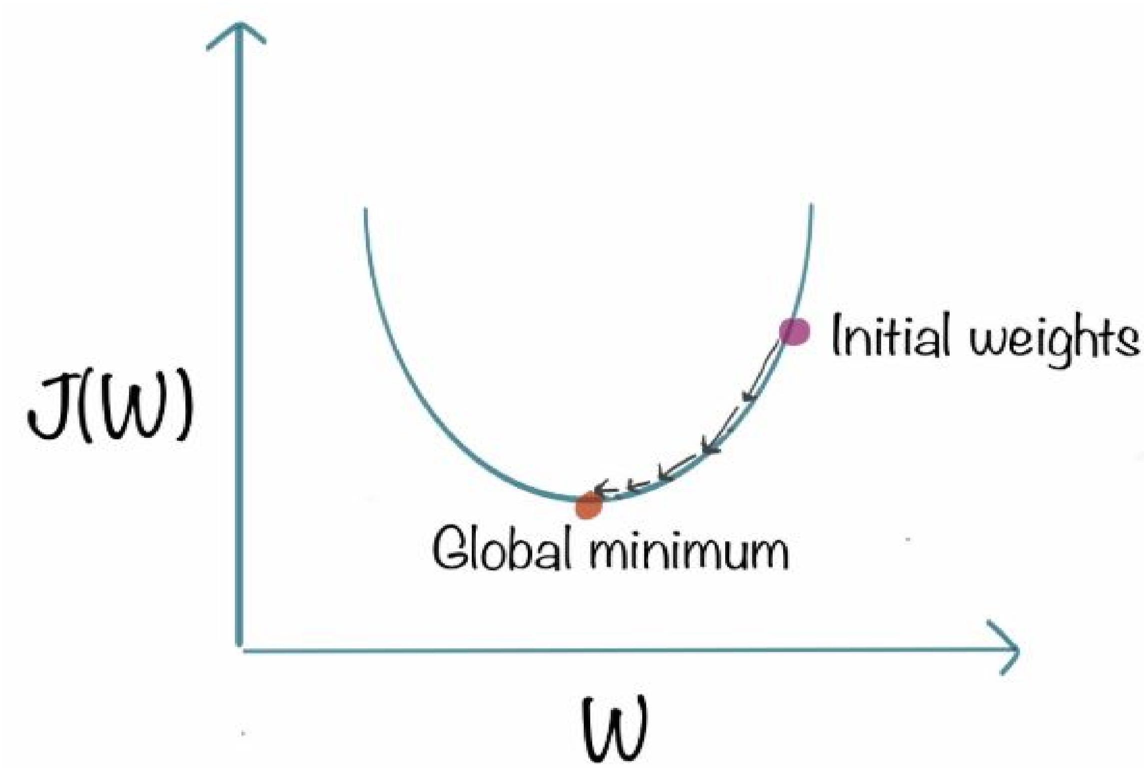
Overview



- in many algorithms we want to compute weights W to minimize cost function $J(W)$

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Basic Idea

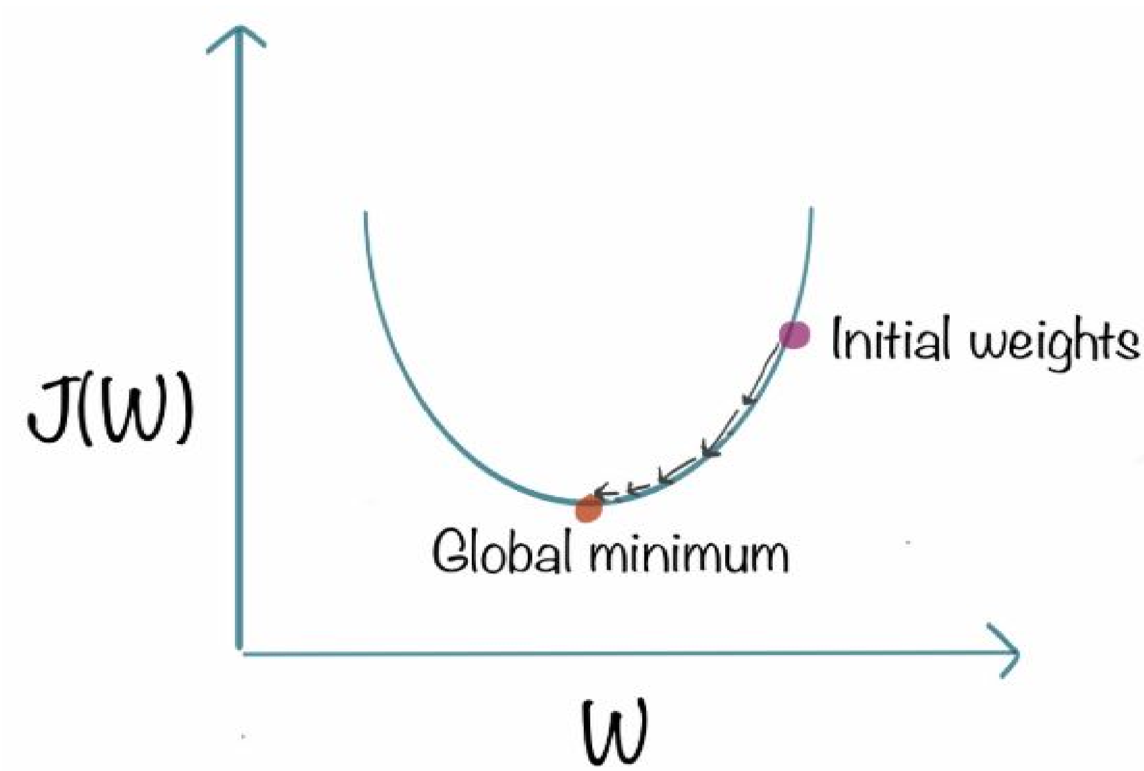


- start with initial weights
- update W to reduce $J(W)$

---

figure reprinted from www.kdnuggets.com with explicit permission of the editor

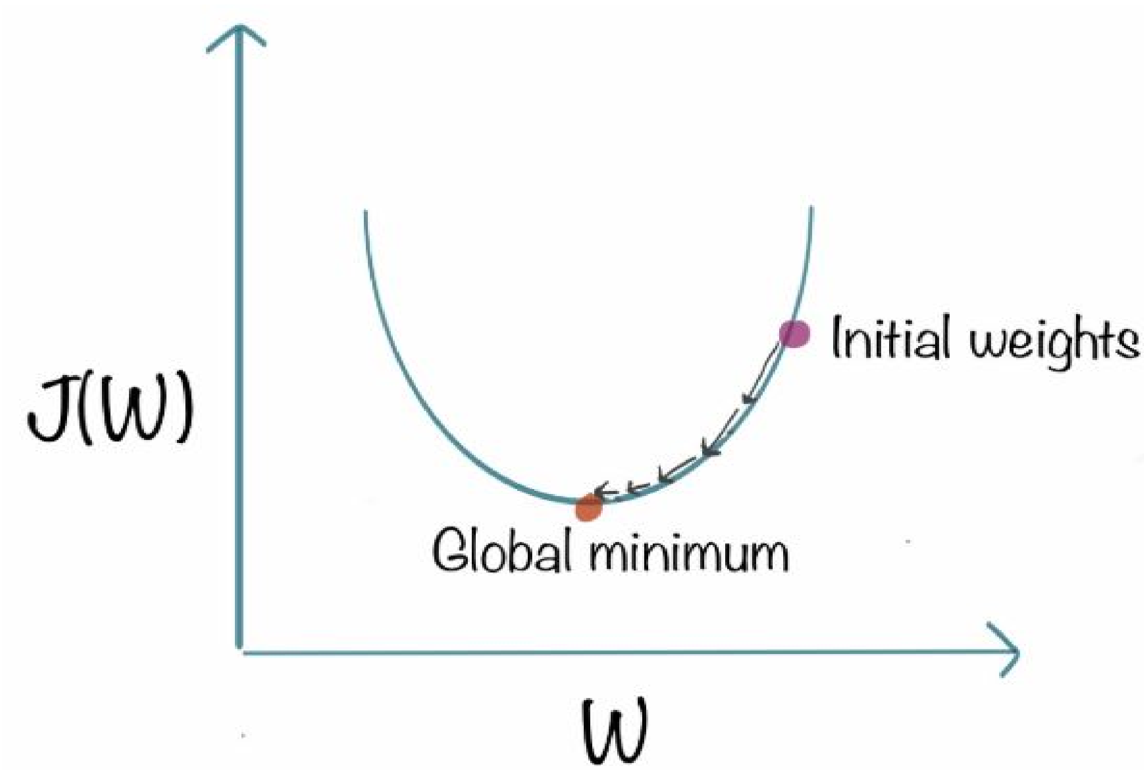
How to Update Weights?



- take steps proportional to the gradient

figure reprinted from www.kdnuggets.com with explicit permission of the editor

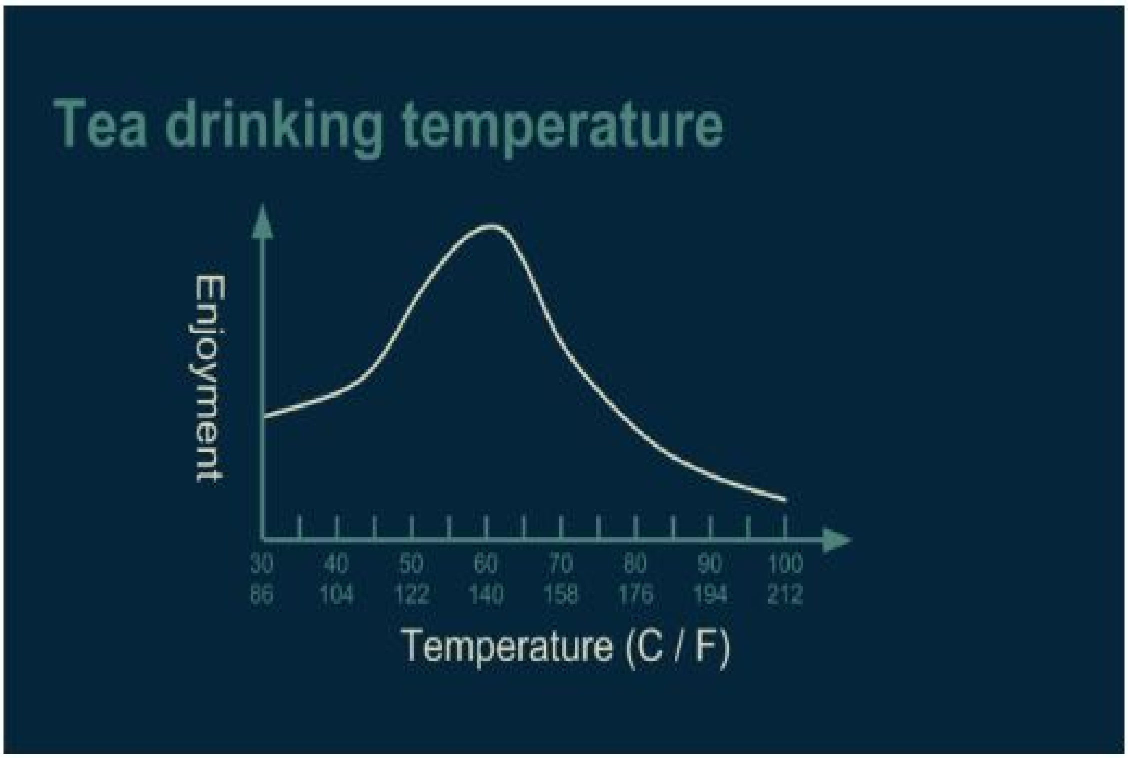
Why Use Gradient?



- cost function $J(W)$ decreases fastest in the direction of negative gradient

figure reprinted from www.kdnuggets.com with explicit permission of the editor

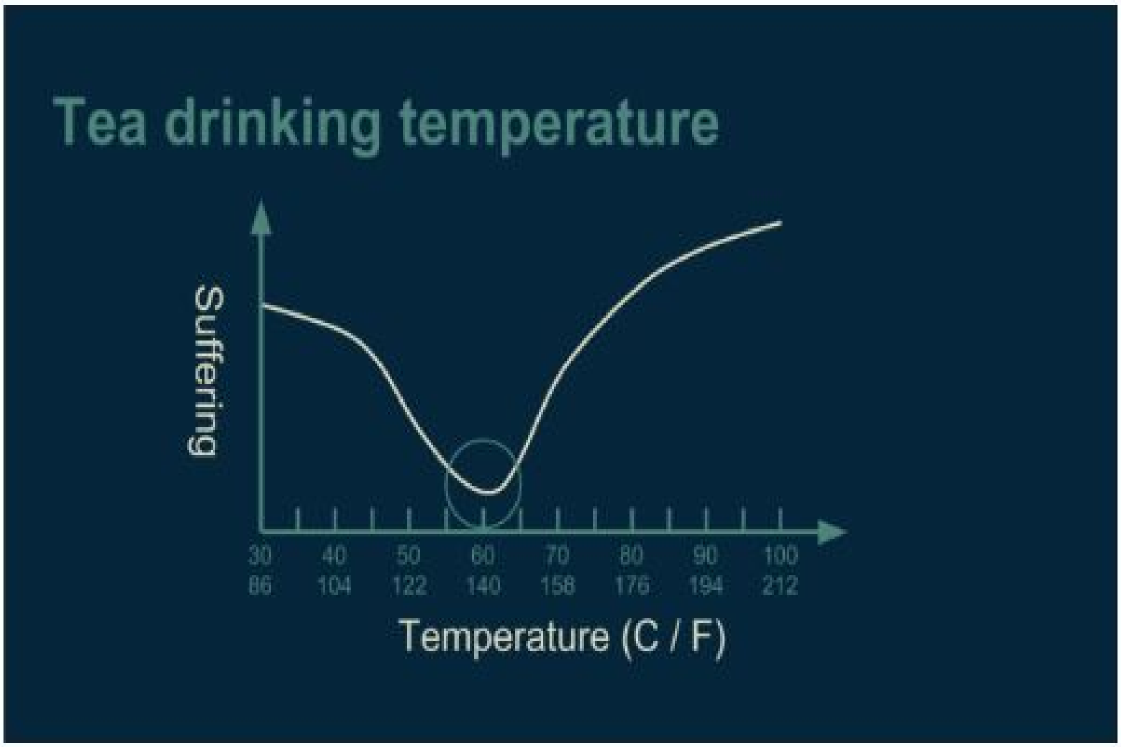
Intuition



- find temperature (”weights”) to maximize ”enjoyment”

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Equivalent Formulation



- find temperature (”weights”) to minimize ”suffering” $J(W)$

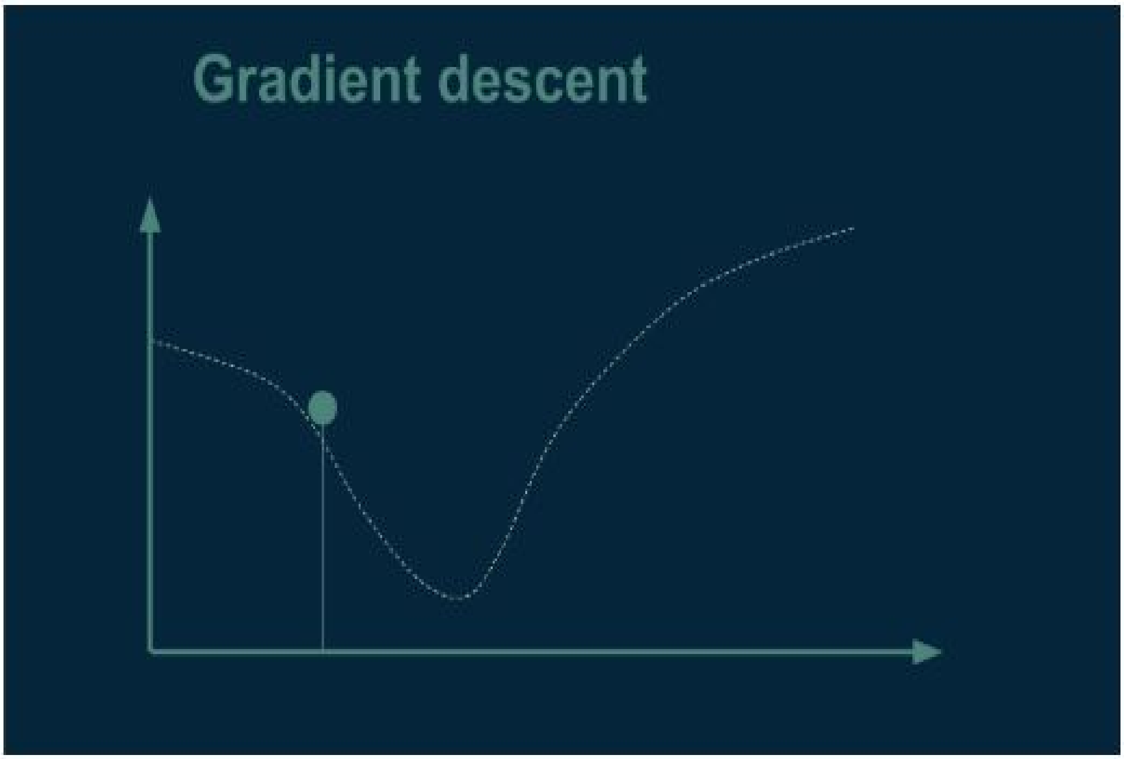
figure reprinted from www.kdnuggets.com with explicit permission of the editor

Exhaustive Search

- can examine "all" values
- this is inefficient

figure reprinted from www.kdnuggets.com with explicit permission of the editor

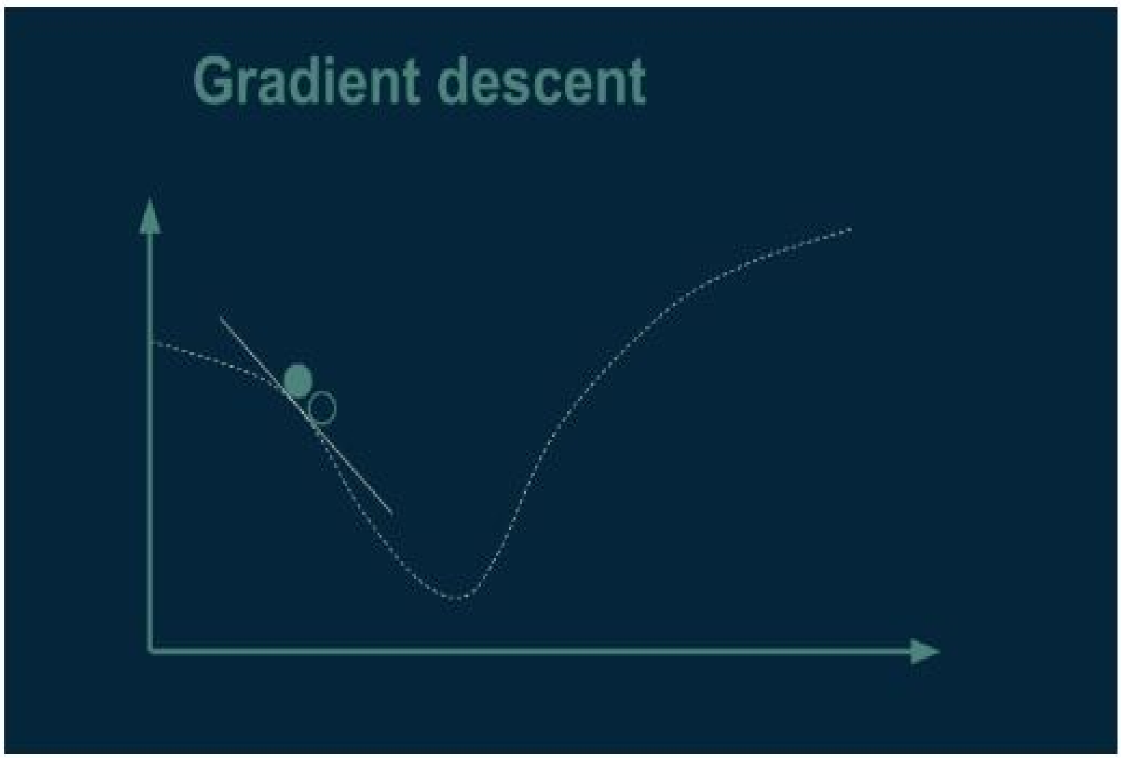
Gradient Descent



- iterative update weights to lower $J(W)$

figure reprinted from www.kdnuggets.com with explicit permission of the editor

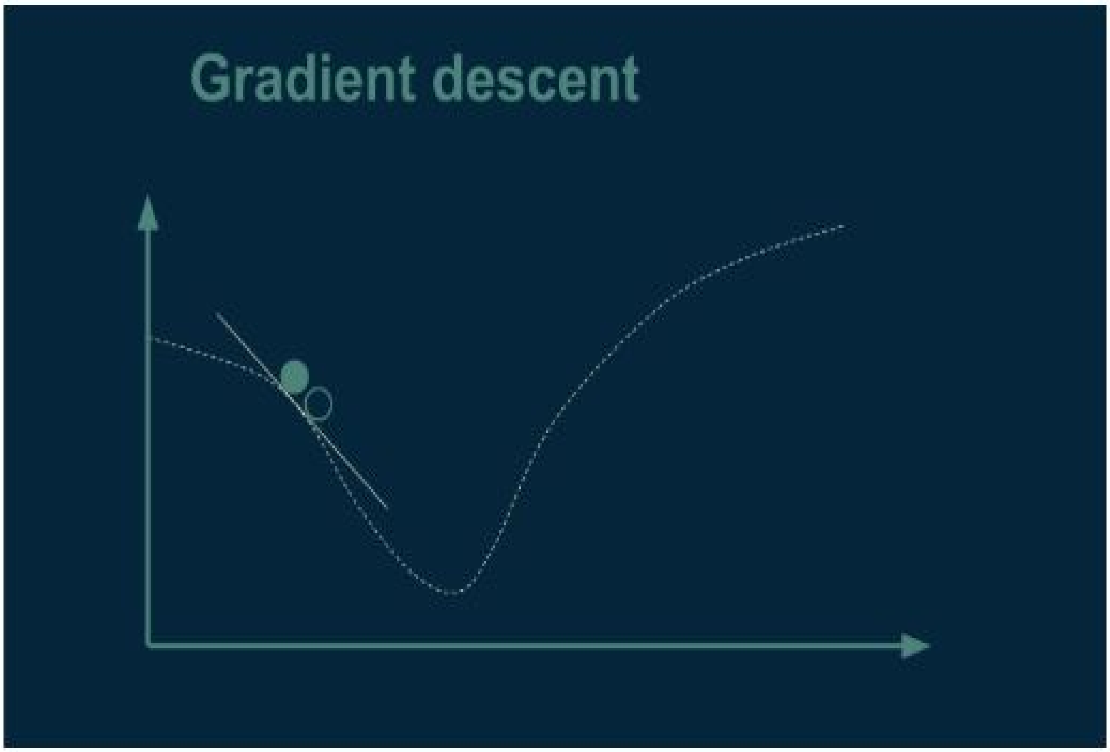
Typical Step



- $J(W)$ is lowered if we move "opposite" gradient

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Typical Step (cont'd)

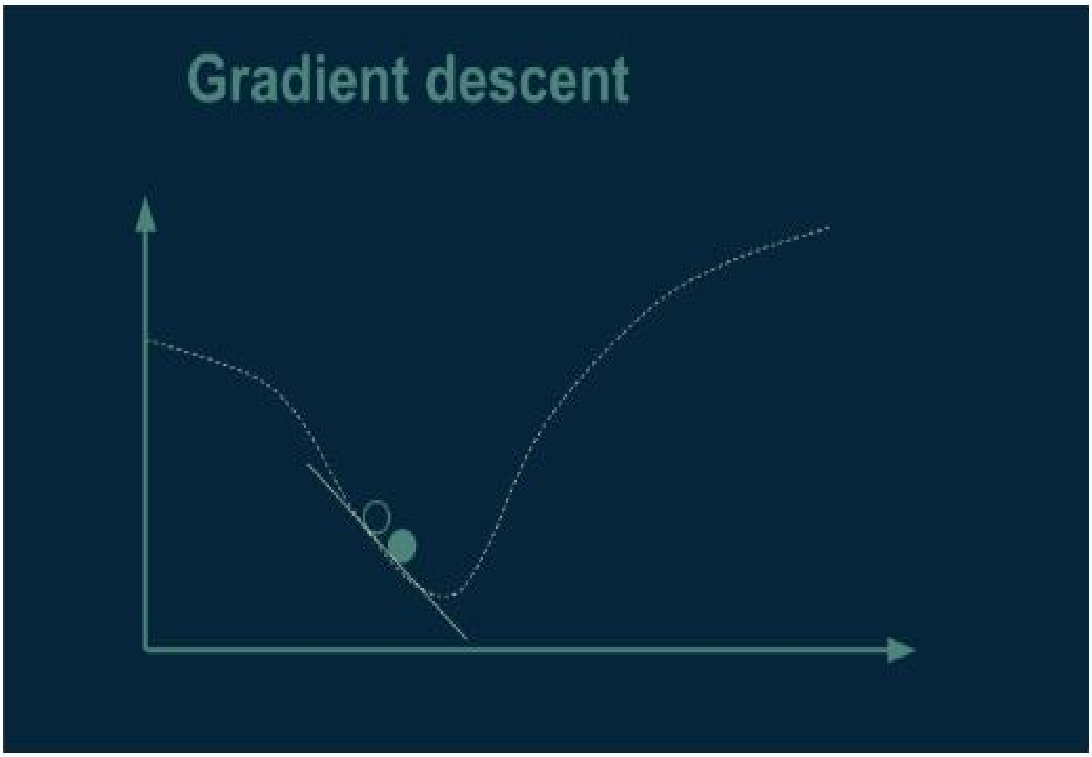


- continue moving "opposite" gradient

---

figure reprinted from www.kdnuggets.com with explicit permission of the editor

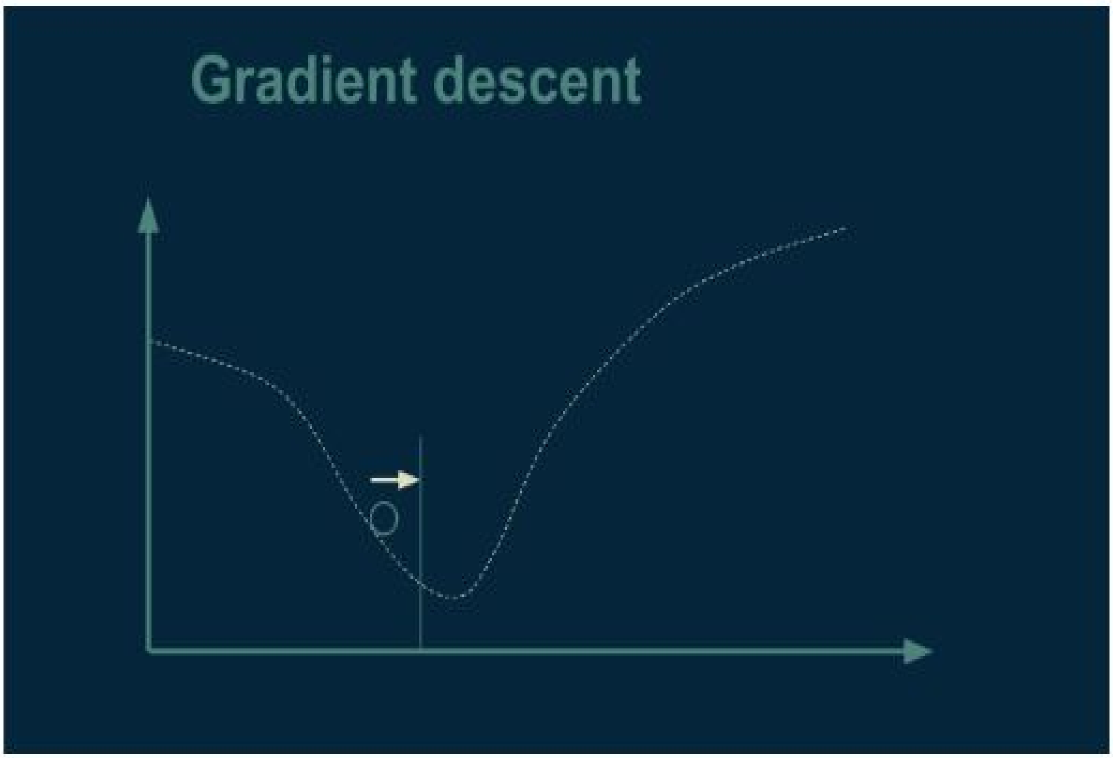
"Speed" of Convergence



- can take larger step for "steeper" slopes

figure reprinted from www.kdnuggets.com with explicit permission of the editor

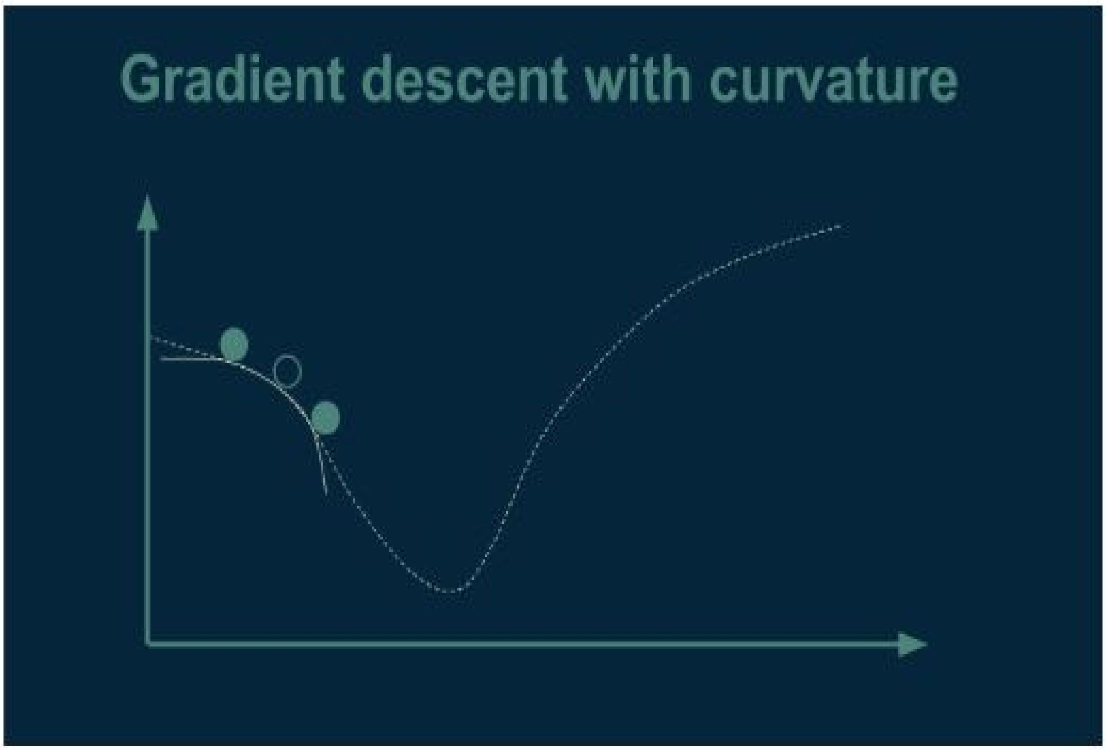
Stopping Criteria



- no ("significant") decrease in $J(W)$

figure reprinted from www.kdnuggets.com with explicit permission of the editor

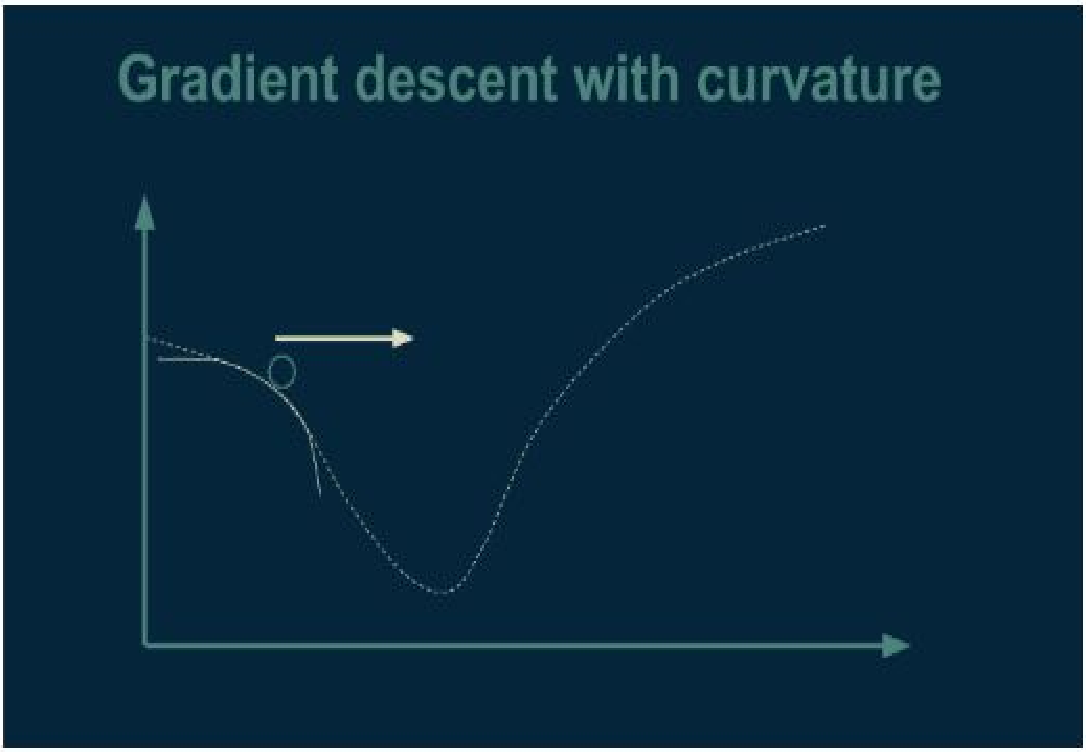
Using Curvature



- can reduce number of steps
- large curvature: big step

figure reprinted from www.kdnuggets.com with explicit permission of the editor

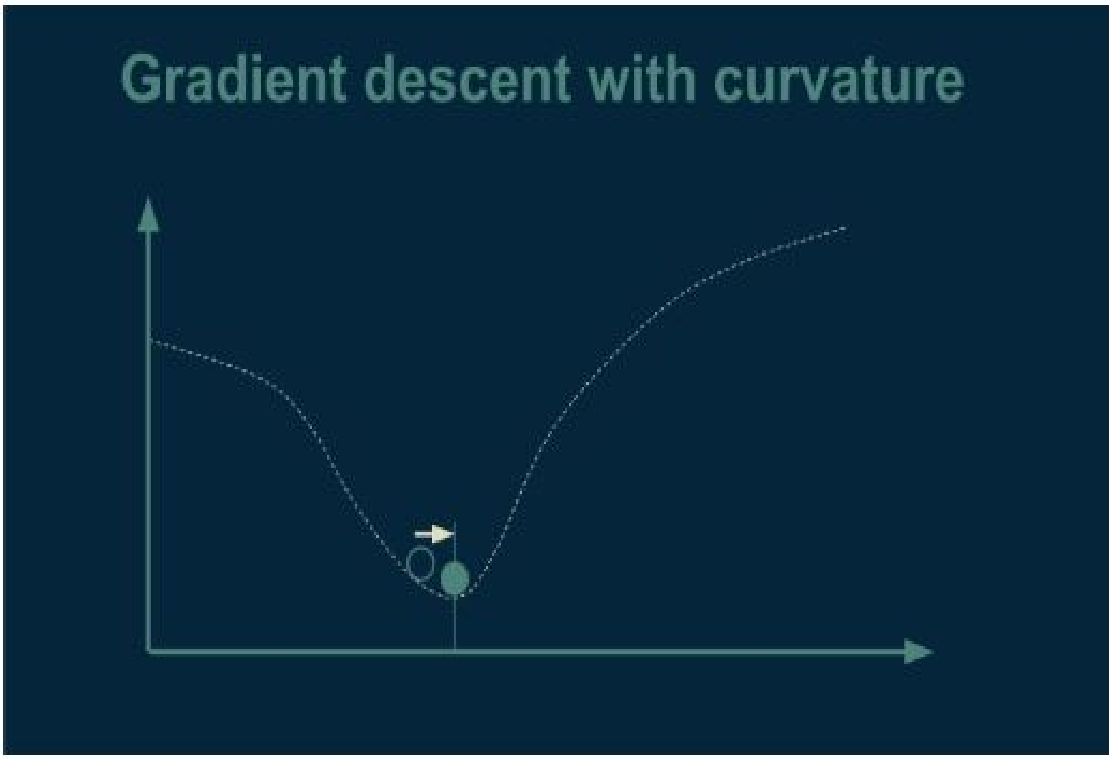
Using Curvature (cont'd)



- small curvature: small step

figure reprinted from www.kdnuggets.com with explicit permission of the editor

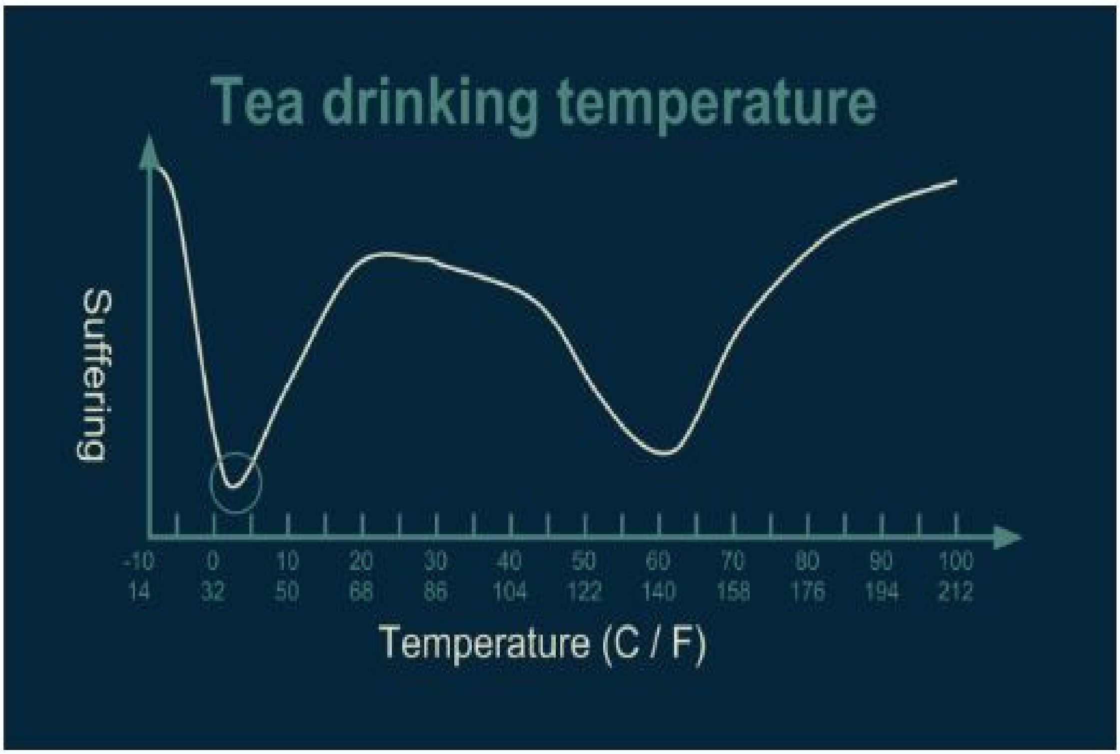
Curvature Trade-off



- use fewer steps
- but more computations

figure reprinted from www.kdnuggets.com with explicit permission of the editor

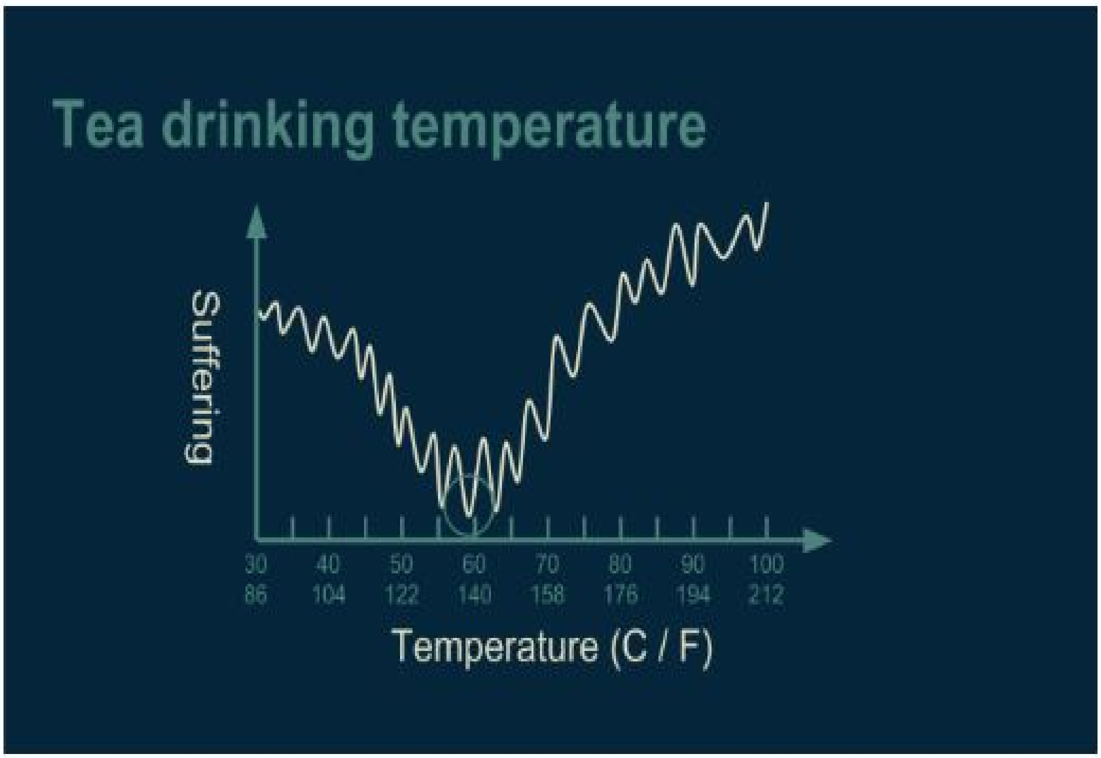
Issue: Local Minimum



- may consider randomization

figure reprinted from www.kdnuggets.com with explicit permission of the editor

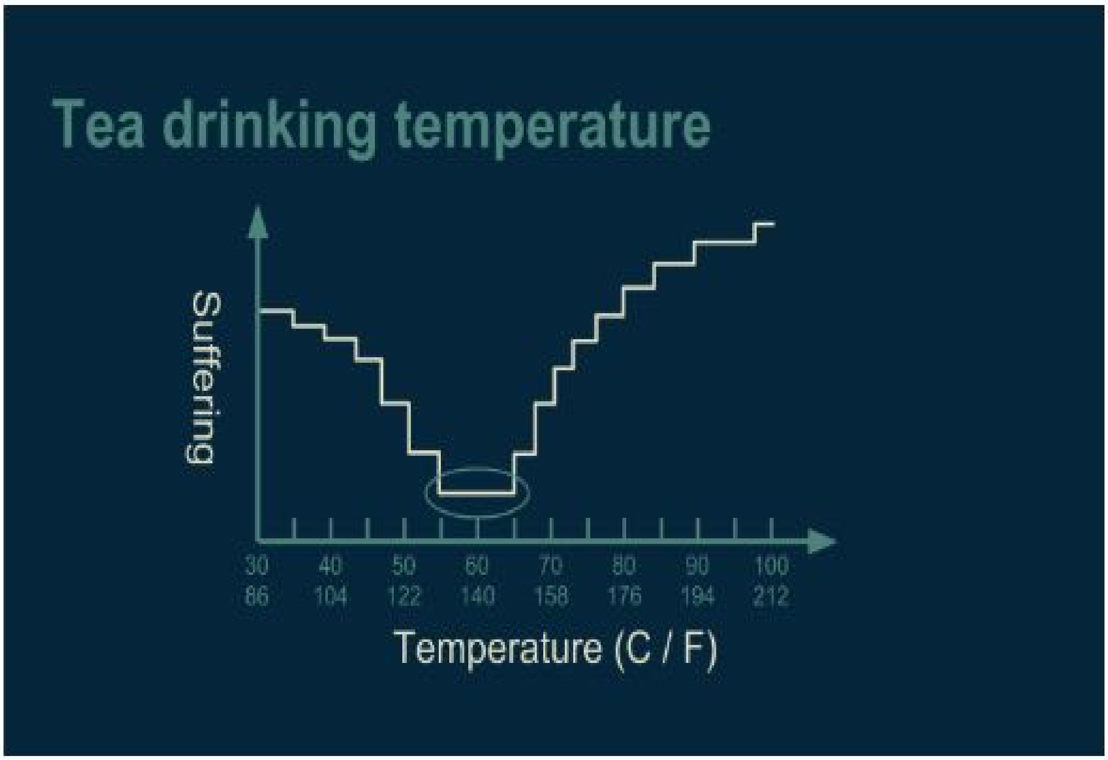
Issue: Noisy Data



- may stop far from optimal

figure reprinted from www.kdnuggets.com with explicit permission of the editor

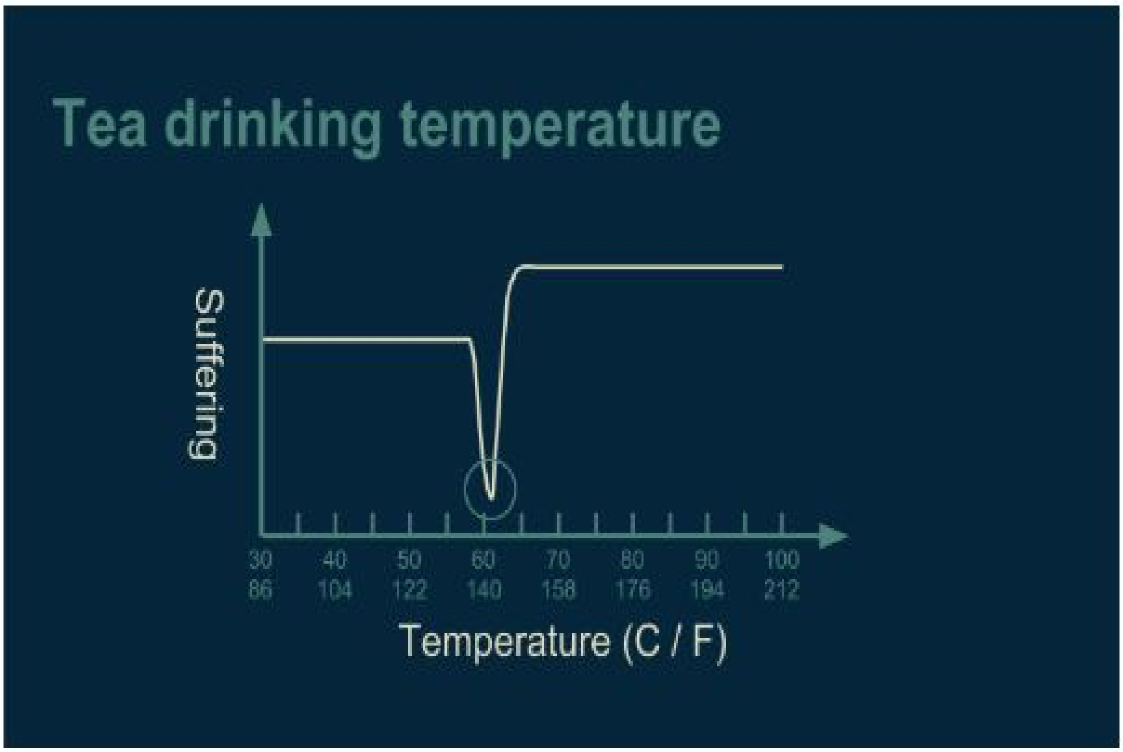
Issue: Zero Gradient



- cannot update weights

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Issue: Discontinuous Cost Function



- may stop far from optimal

figure reprinted from www.kdnuggets.com with explicit permission of the editor

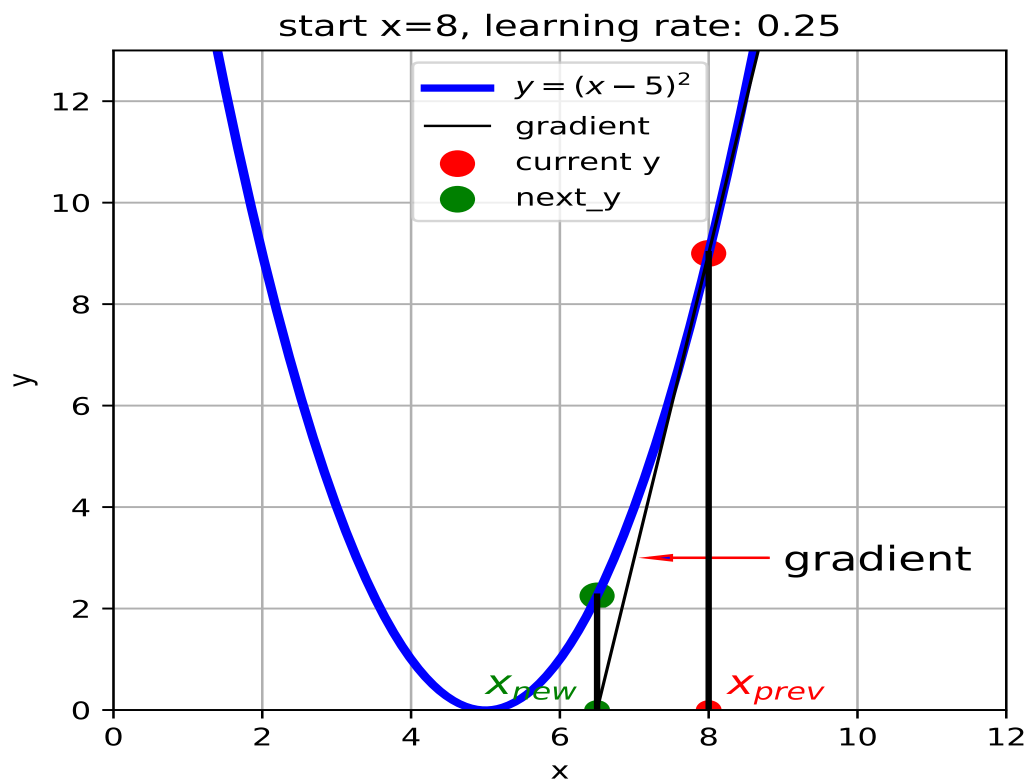
The Algorithm

- compute W to minimize $f(x)$
- choose initial weights W_0 and learning rate α
- for iteration i update weights:

$$W_i = W_{i-1} - \alpha \cdot \partial f(W_{i-1})$$

- repeat iterations until changes in W do not (significantly) change $f(\cdot)$

Geometric Interpretation



- iterative optimization
- direction to minimize $f(x)$

Python Code

```
import numpy as np
x  = np.linspace(0, 10, 1000)
y  = (x - 5)**2
df = lambda x: 2*(x-5)

rate   =0.25; precision      = 0.001
next_x = 8;   max_iterations = 100

for i in range(max_iterations):
    cur_x  = next_x
    next_x = cur_x - rate * df(cur_x)
    step   = next_x - cur_x
    print("Iteration:",i+1,"x= ",next_x)
    if abs(step) <= precision:
        break

print("Minimum at", next_x, ', ',
       i+1, ' iterations')
```

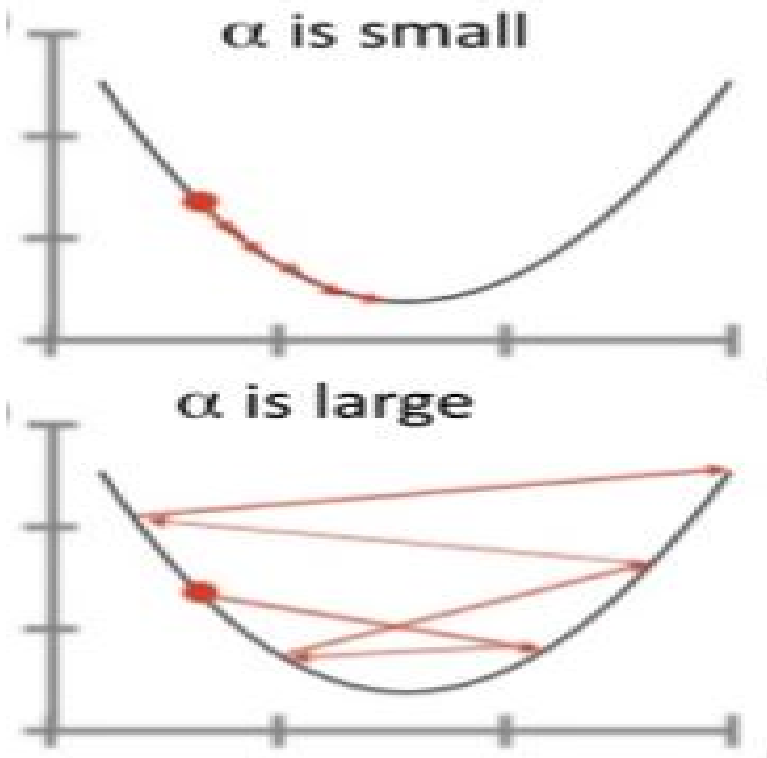
Execution Details

- for iteration 1:

$$cur\_x = next\_x = 8$$
$$df\left(cur\_x\right) = 2 * (8 - 5) = 6$$
$$next\_x = cur\_x - rate * df\left(cur\_x\right)$$
$$= 8 - 0.25 * 6 = 6.5$$

```
Iteration:  1   x =  6.5
Iteration:  2   x =  5.75
Iteration:  3   x =  5.375
-----------------------
-----------------------
Iteration:  10  x =  5.0029296875
Iteration:  11  x =  5.00146484375
Iteration:  12  x =  5.000732421875
Minimum at 5.000732421875, 12  iterations
```

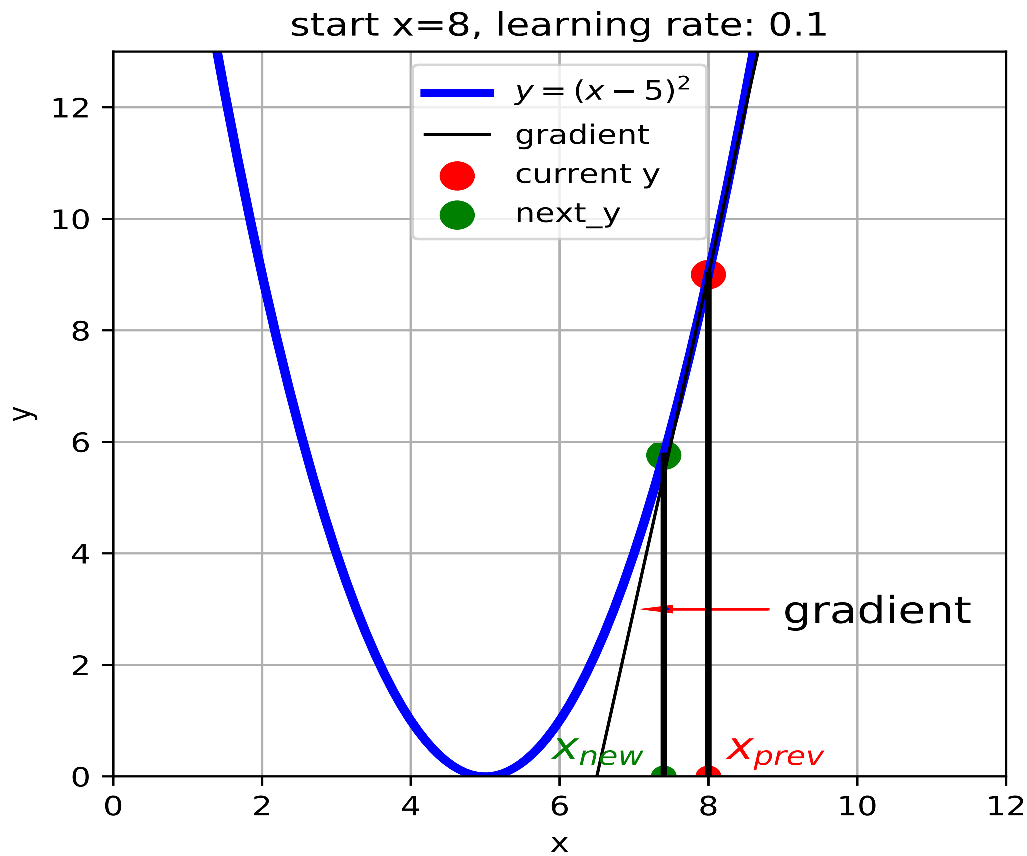
Choosing the Learning Rate



- small α - slow convergence
- large α - possible oscillations

figure reprinted from www.kdnuggets.com with explicit permission of the editor

Effect of Decreasing Rate



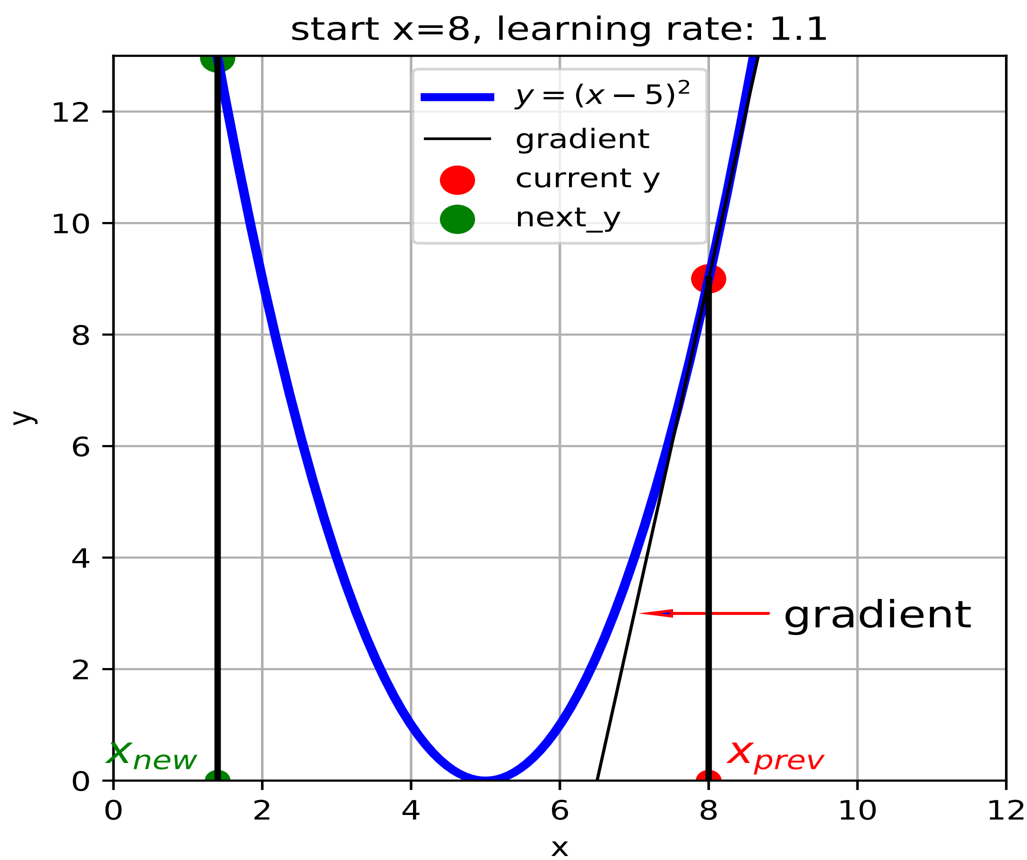
- slower rate - more iterations

Effect of Decreasing Rate (cont'd)

- rate = 0.25, next_x = 8

```
Iteration:  1   x =  7.4
Iteration:  2   x =  6.92
Iteration:  3   x =  6.536
------------------------
------------------------
Iteration:  29  x =  5.0046422751473205
Iteration:  30  x =  5.003713820117857
Minimum at 5.003713820117857, 30  iterations
```

Effect of Increasing Rate



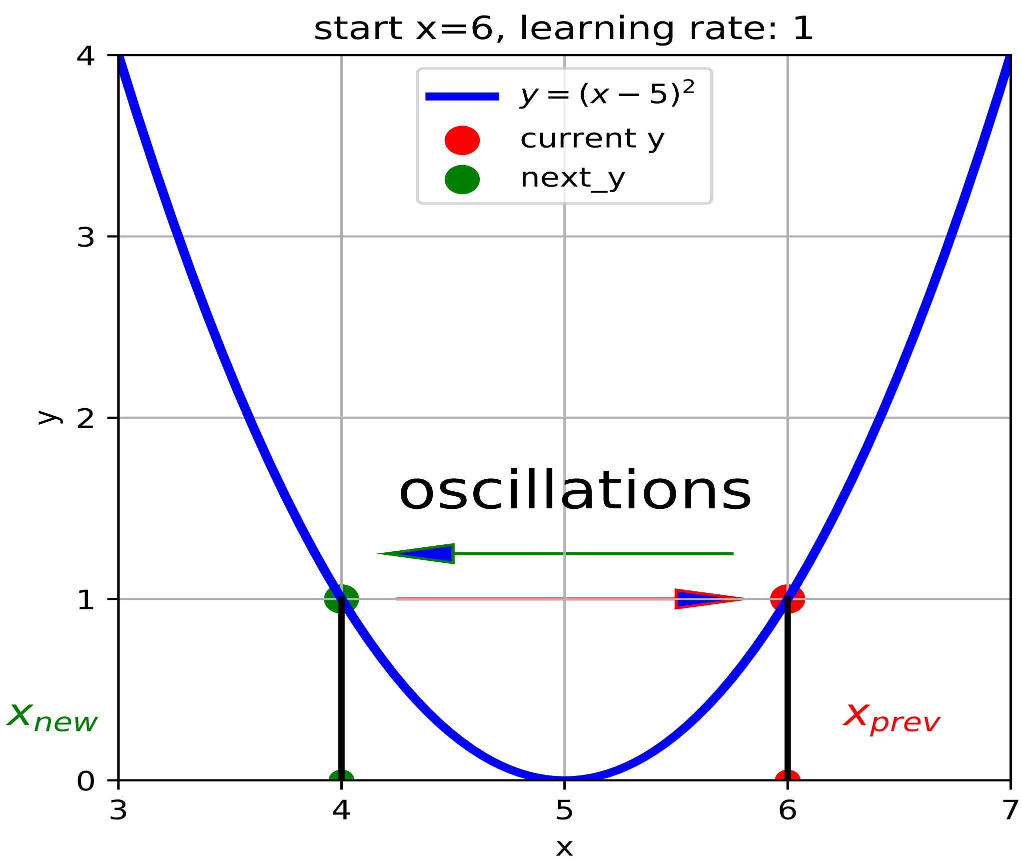
- may fail to converge

Effect of Increasing Rate (cont'd)

- rate = 1.1, next_x = 8
- may fail to converge

```
Iteration:  1   x =  1.3999999999999995
Iteration:  2   x =  9.32
Iteration:  3   x =  -0.1840000000000105
---------------------
---------------------
Iteration:  99  x =  -207044931.30503917
Iteration:  100 x =  248453928.566047
Minimum at 248453928.566047, 100  iterations
```

Oscillations



- never converges

Example of Oscillations

- take $rate = 1, next\_x = 6$
- iteration 1:

$$cur\_x = next\_x = 6$$

$$df(cur\_x) = 2 * (6 - 5) = 2$$

$$next\_x = cur\_x - rate * df(cur\_x)$$

$$= 6 - 1 * 2 = 4$$

- iteration 2:

$$cur\_x = next\_x = 4$$

$$df(cur\_x) = 2 * (4 - 5) = -2$$

$$next\_x = cur\_x - rate * df(cur\_x)$$

$$= 4 - 1 * (-2) = 6$$

Notes on Gradient Descent

- slow - each iteration requires to examine all samples
- some variants - stochastic gradient descent (examine some points)
- how to compute rate?

1. constant
2. decrease with updates

Concepts Check:

(a) optimization by iterations

(b) gradient

(c) curvature

(d) stopping criteria

(e) learning rate

(f) oscillations