
DATA 442: Neural Networks & Deep Learning

Dan Runfola – danr@wm.edu

icss.wm.edu/data442/



Summary

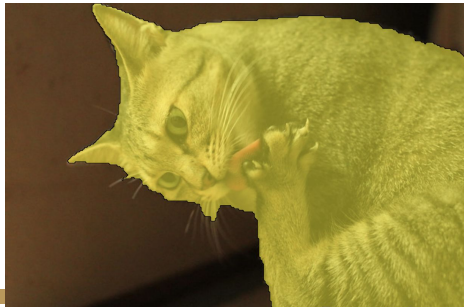
Total Loss=

$$\sum_{i=1}^N \{(x_i, y_i)\}$$

$$\frac{1}{N} \sum_i^N Loss_i(f(x_i, W), y_i) + \lambda R(W)$$

def predict(image, W):
 return(W*image)

Cat	3.2	1.3	2.2
Car	5.1	4.9	2.5
Frog	-1.7	2.0	-3.1



$$L_i = -\log\left(\frac{e_k^s}{\sum_{j=1}^J e_j^s}\right)$$

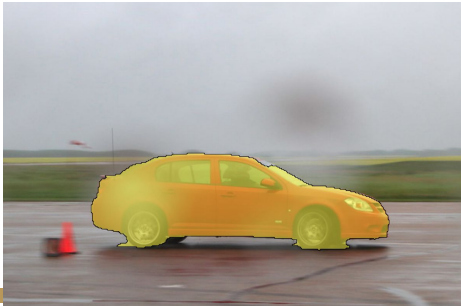
Summary

Total Loss=

$$\sum_{i=1}^{N=3} \{(x_i, y_i)\} \quad \frac{1}{N} \sum_i^N Loss_i(f(x_i, W), y_i) + \lambda R(W)$$

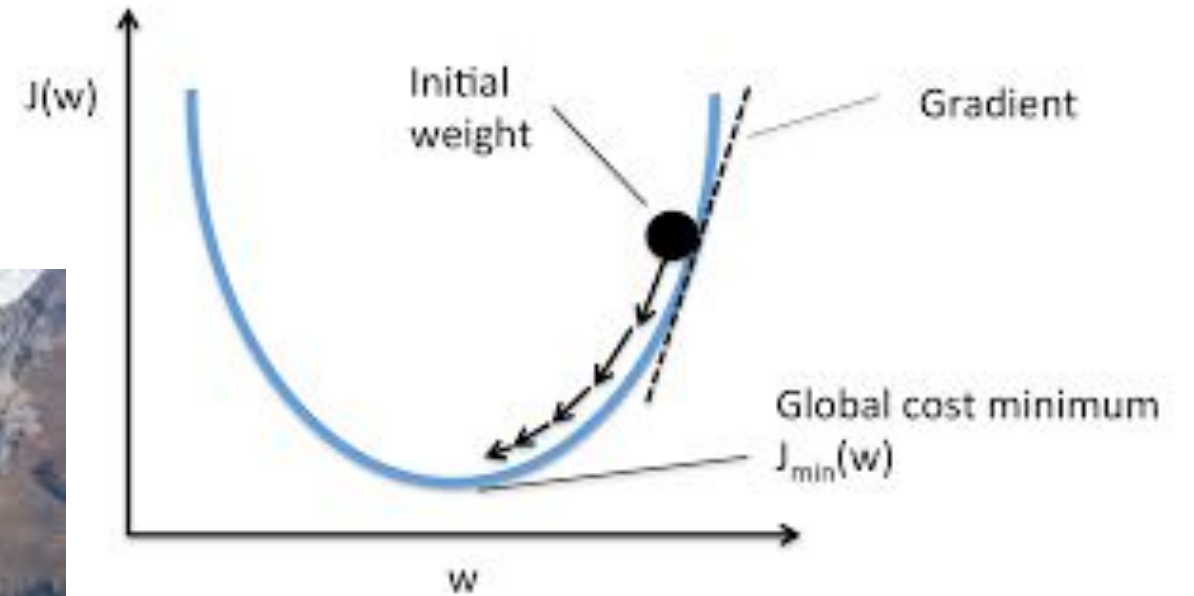
```
def predict(image, W):  
    return(W*image)
```

Cat	3.2	1.3	2.2
Car	5.1	4.9	2.5
Frog	-1.7	2.0	-3.1

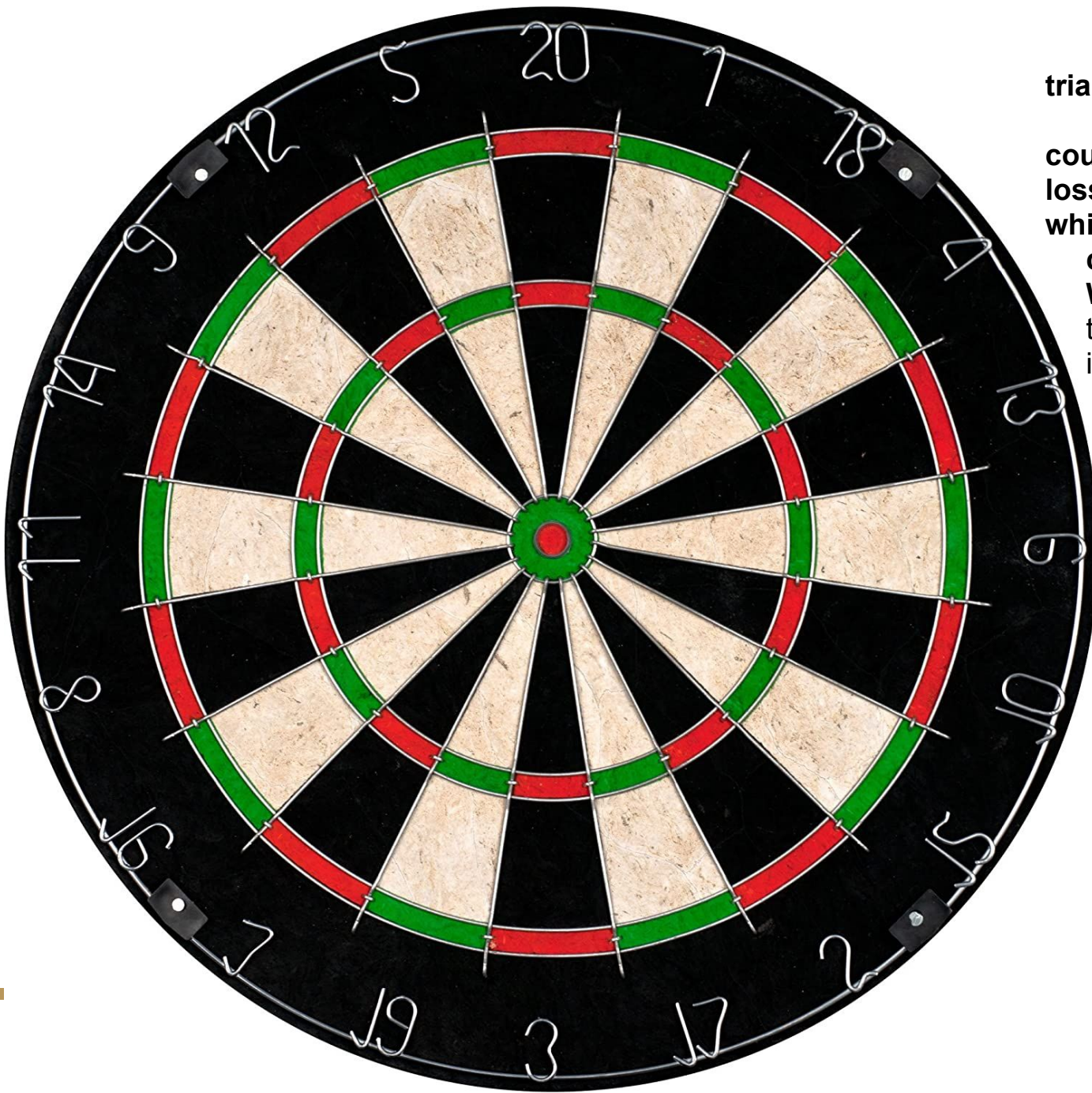


$$L_i = -\log\left(\frac{e_k^s}{\sum_{j=1}^J e_j^s}\right)$$

Optimization



ITY CENTER OF EXCELLENCE



```
trials = 100
```

```
count = 0
```

```
loss = 99999
```

```
while count < trials:
```

```
    count = count + 1
```

```
    W = np.random.randn(10,3072)
```

```
    totalLoss = svmClassifier(X_train, y_train, W, e=1, l=1)
```

```
    if totalLoss < loss:
```

```
        loss = totalLoss
```

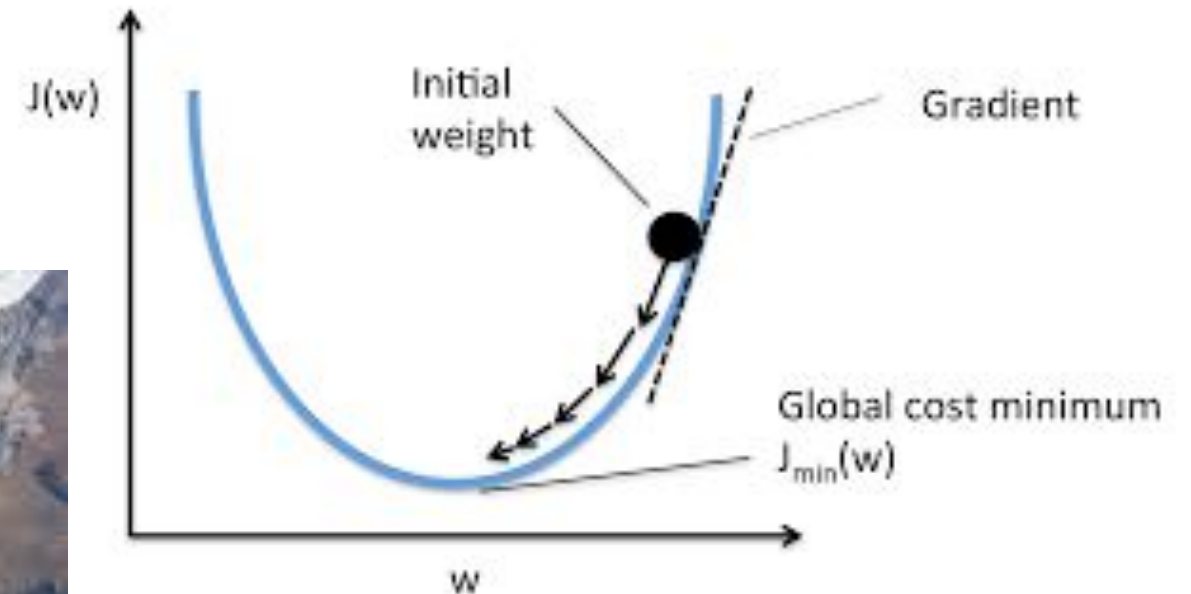
```
        finalWeights = W
```

ACCELERATING

EFFICIENCY

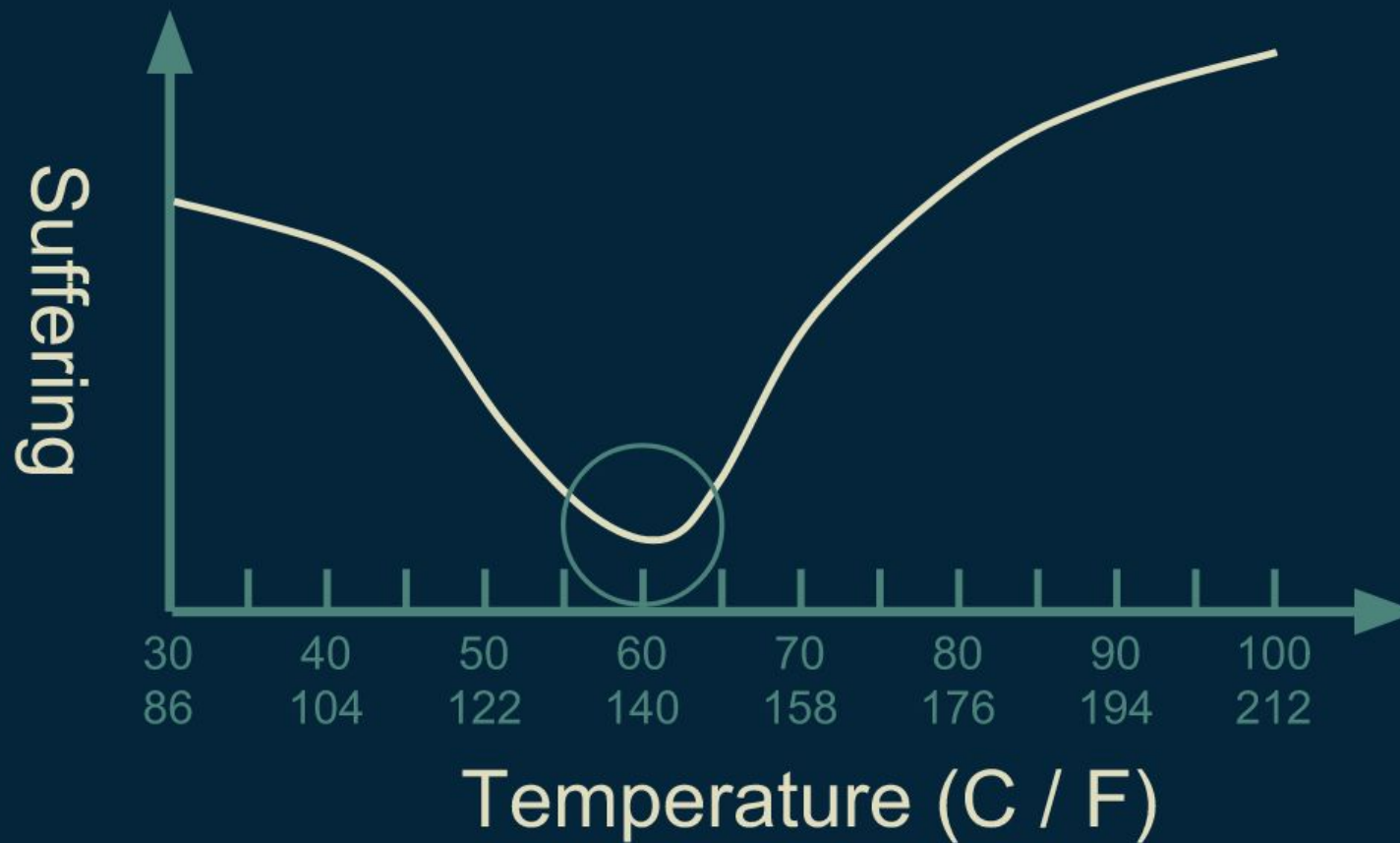
OF EXCELLENCE

Optimization



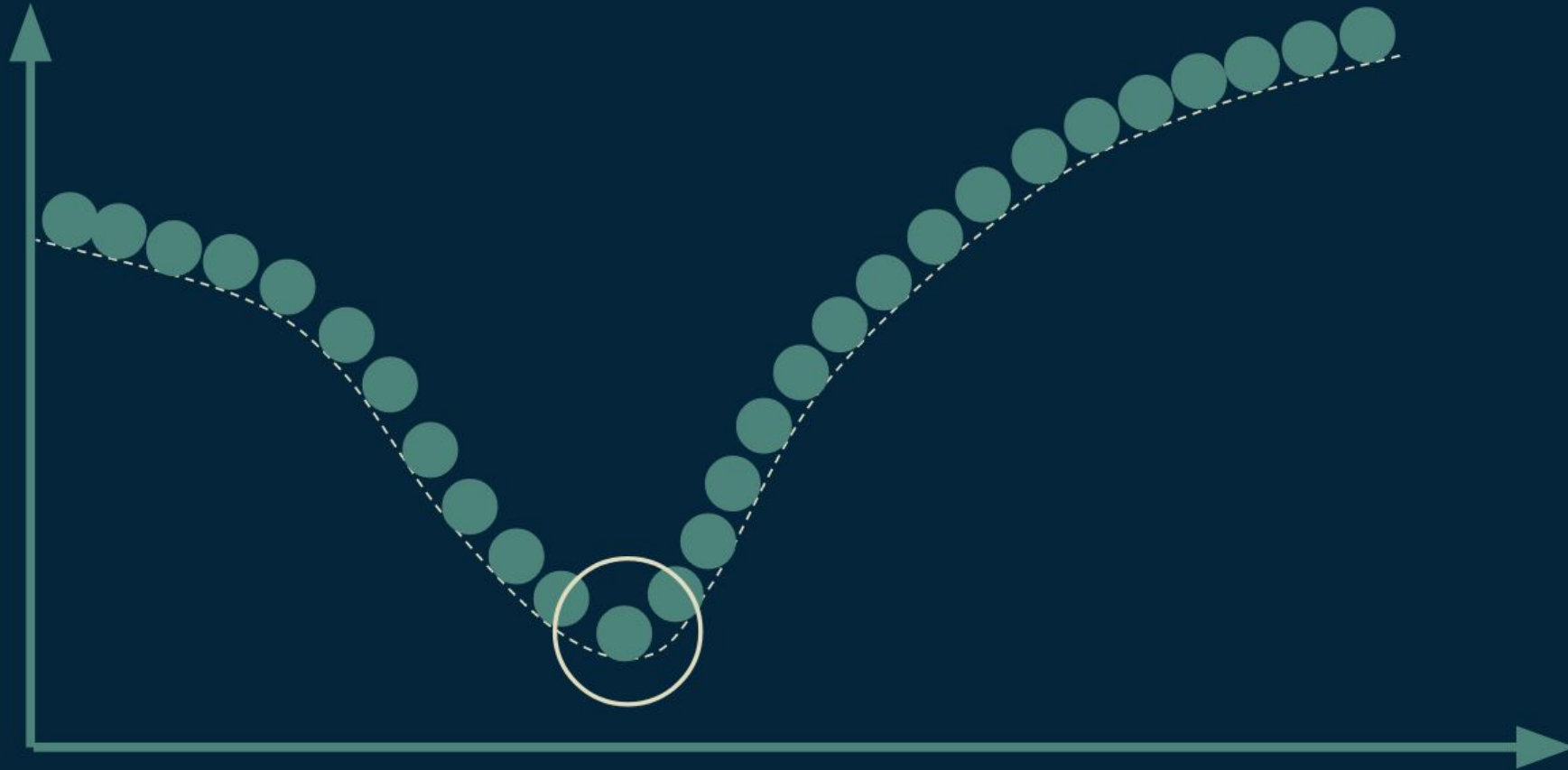
ITY CENTER OF EXCELLENCE

Tea drinking temperature

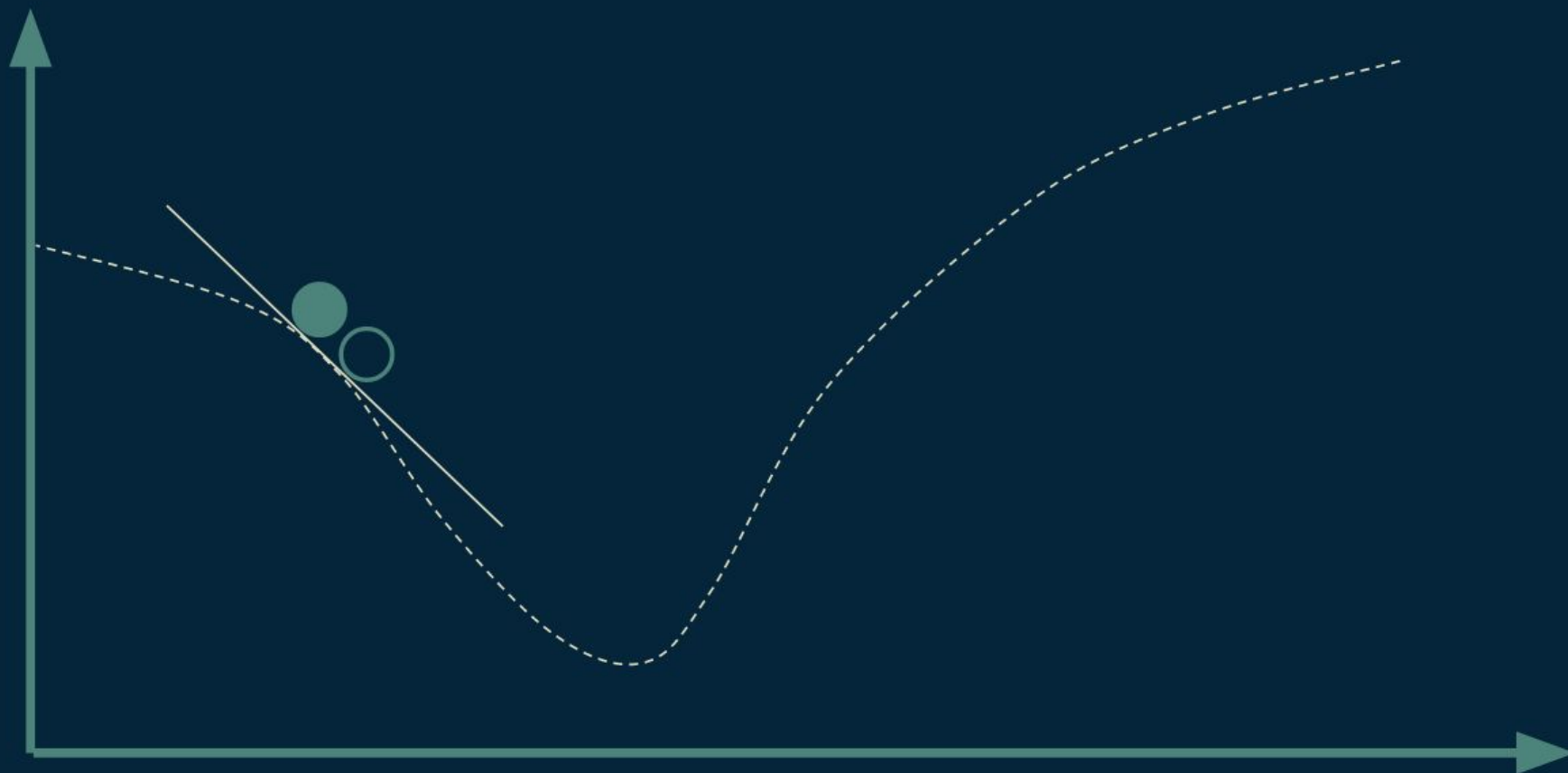


Awesome example from https://brohrer.github.io/how_optimization_works_1.html

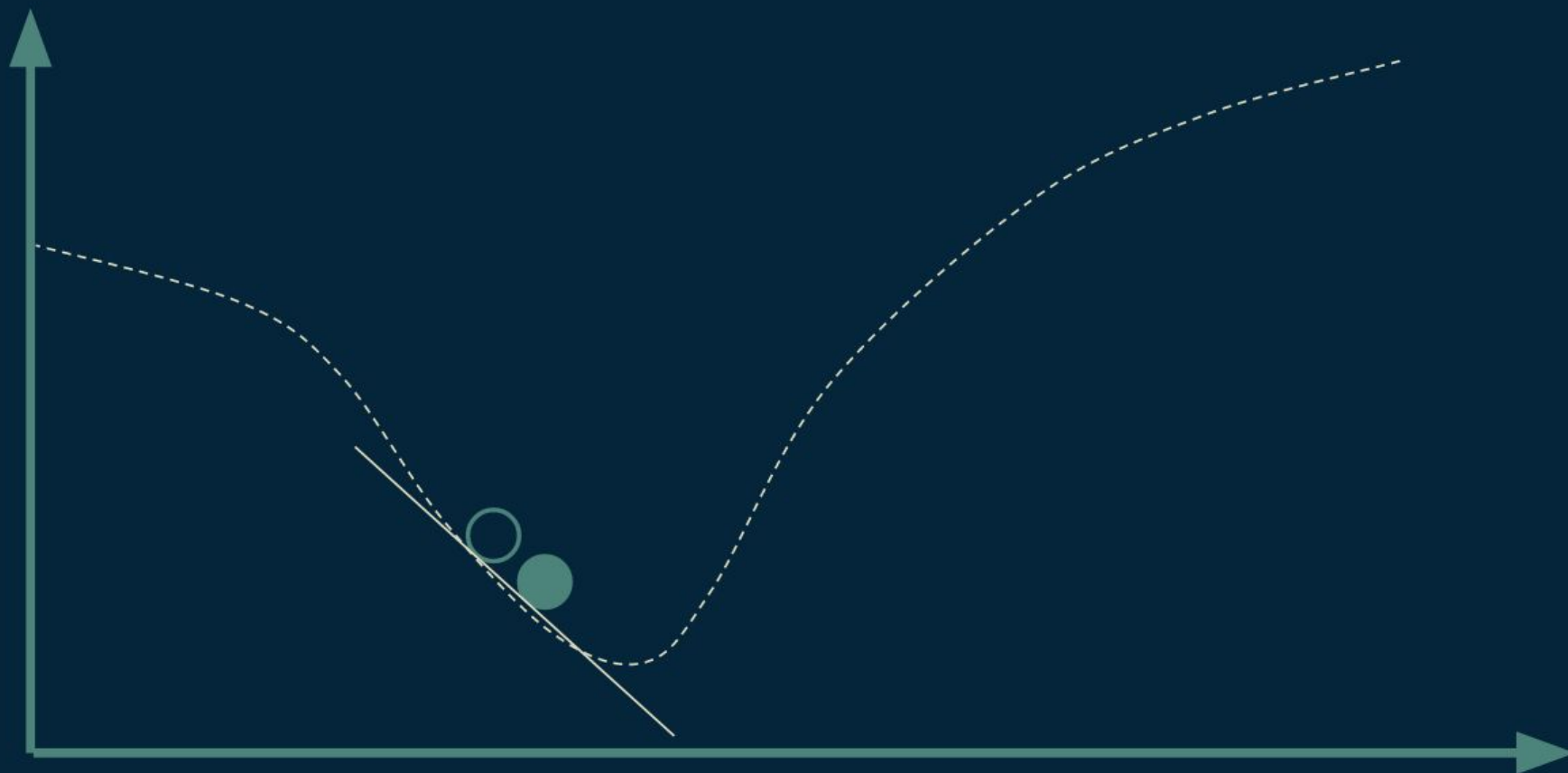
Exhaustive search



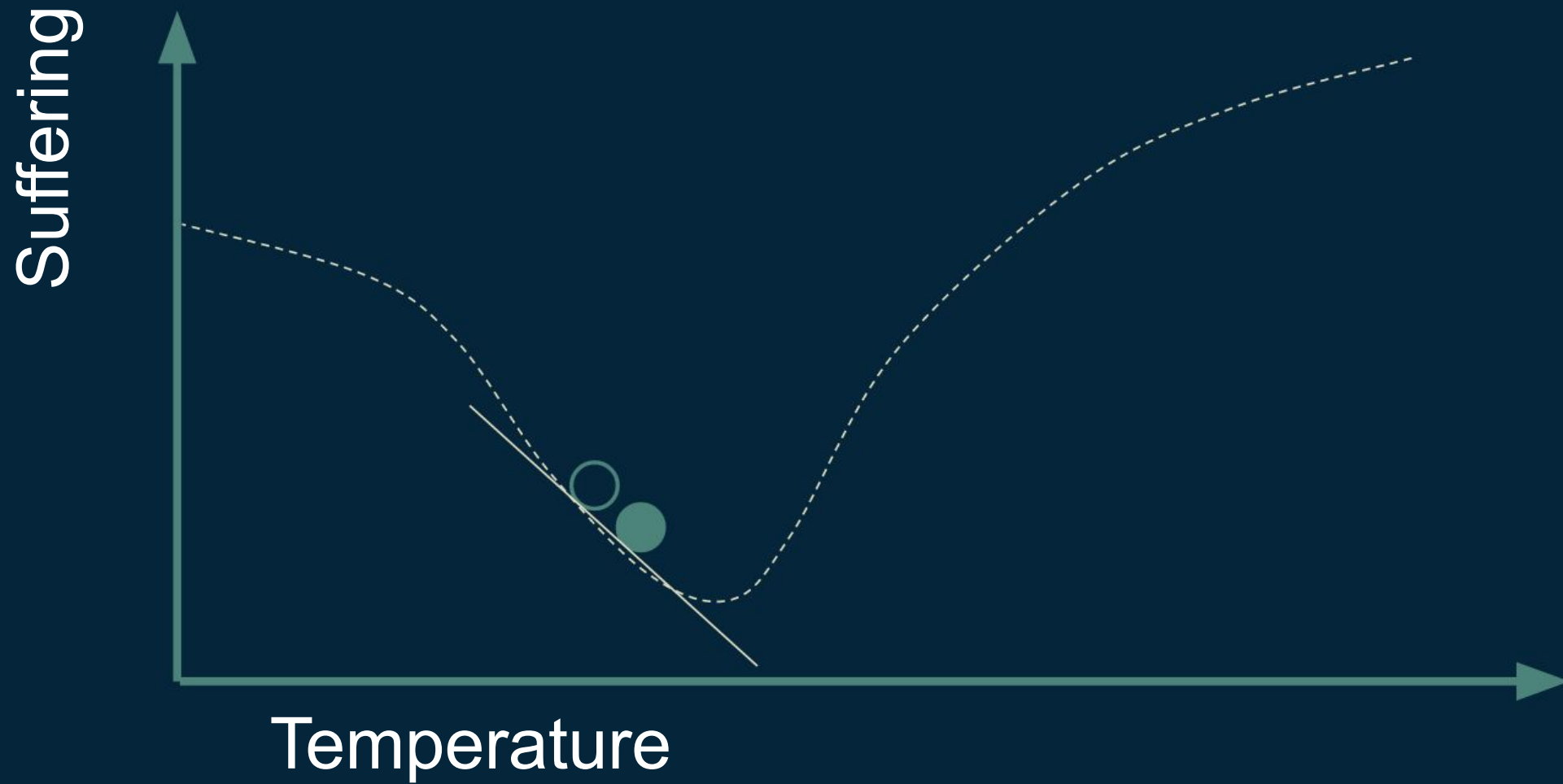
Gradient descent



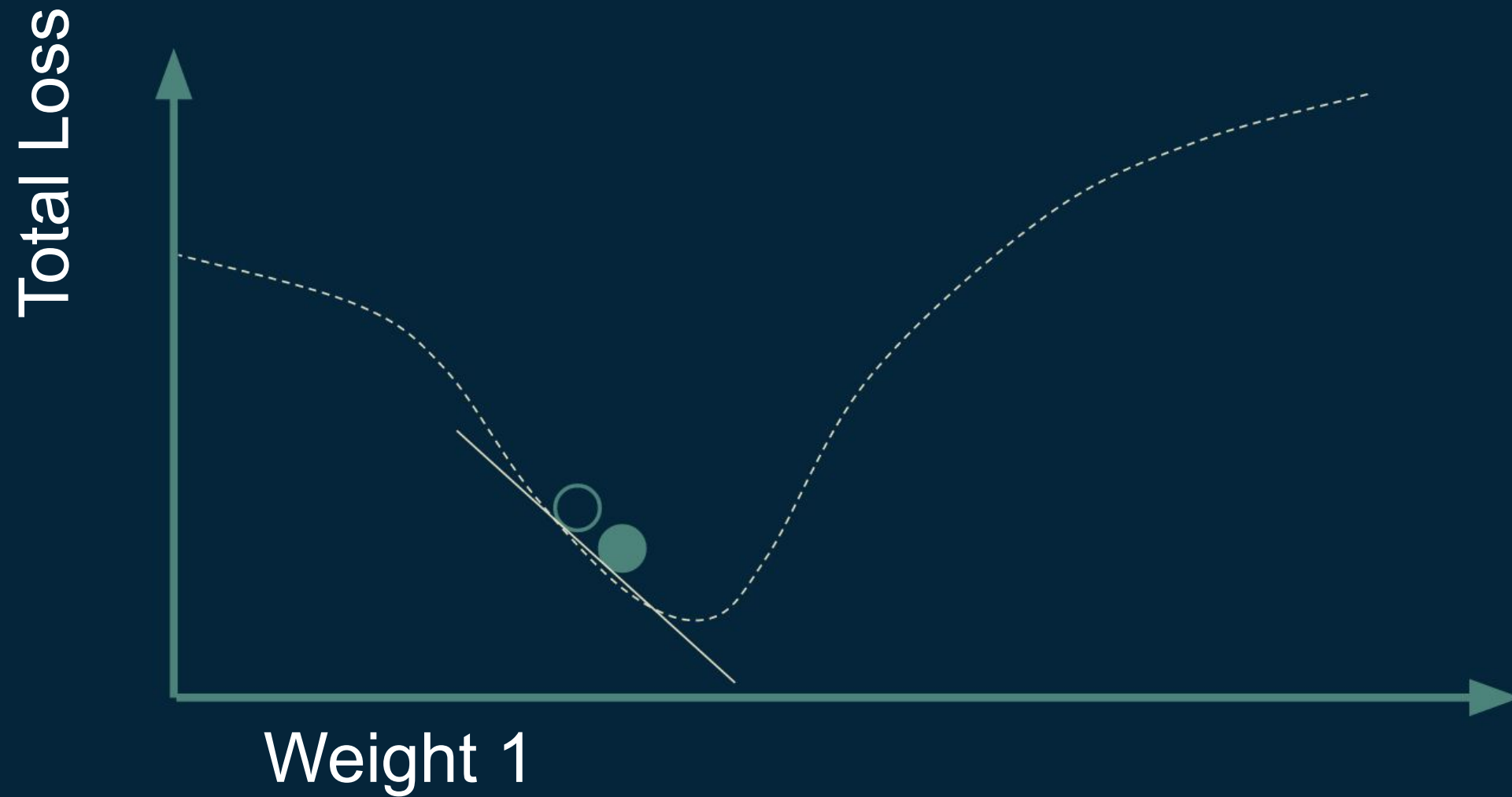
Gradient descent



Gradient descent



Gradient descent

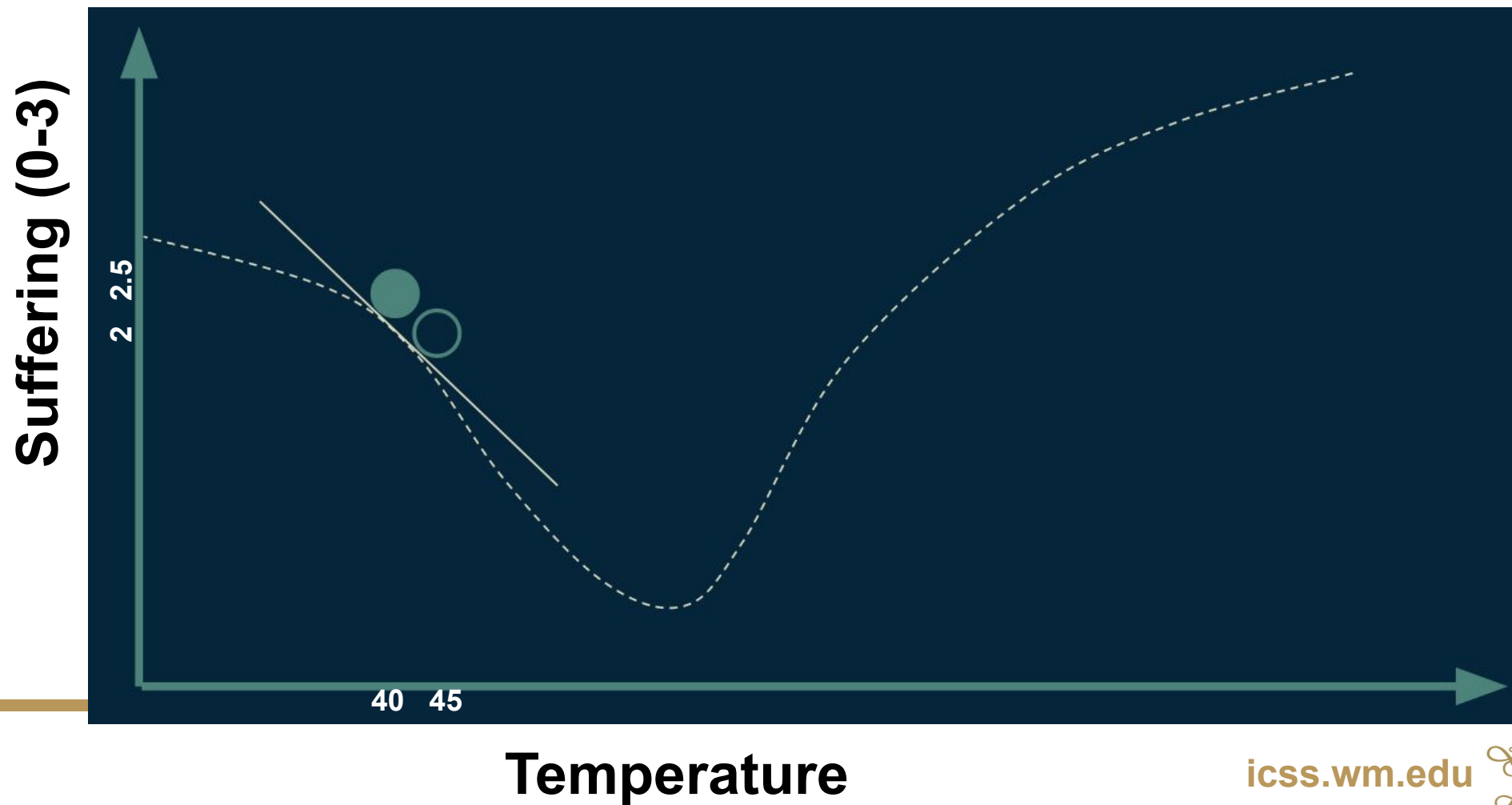


Gradient Descent

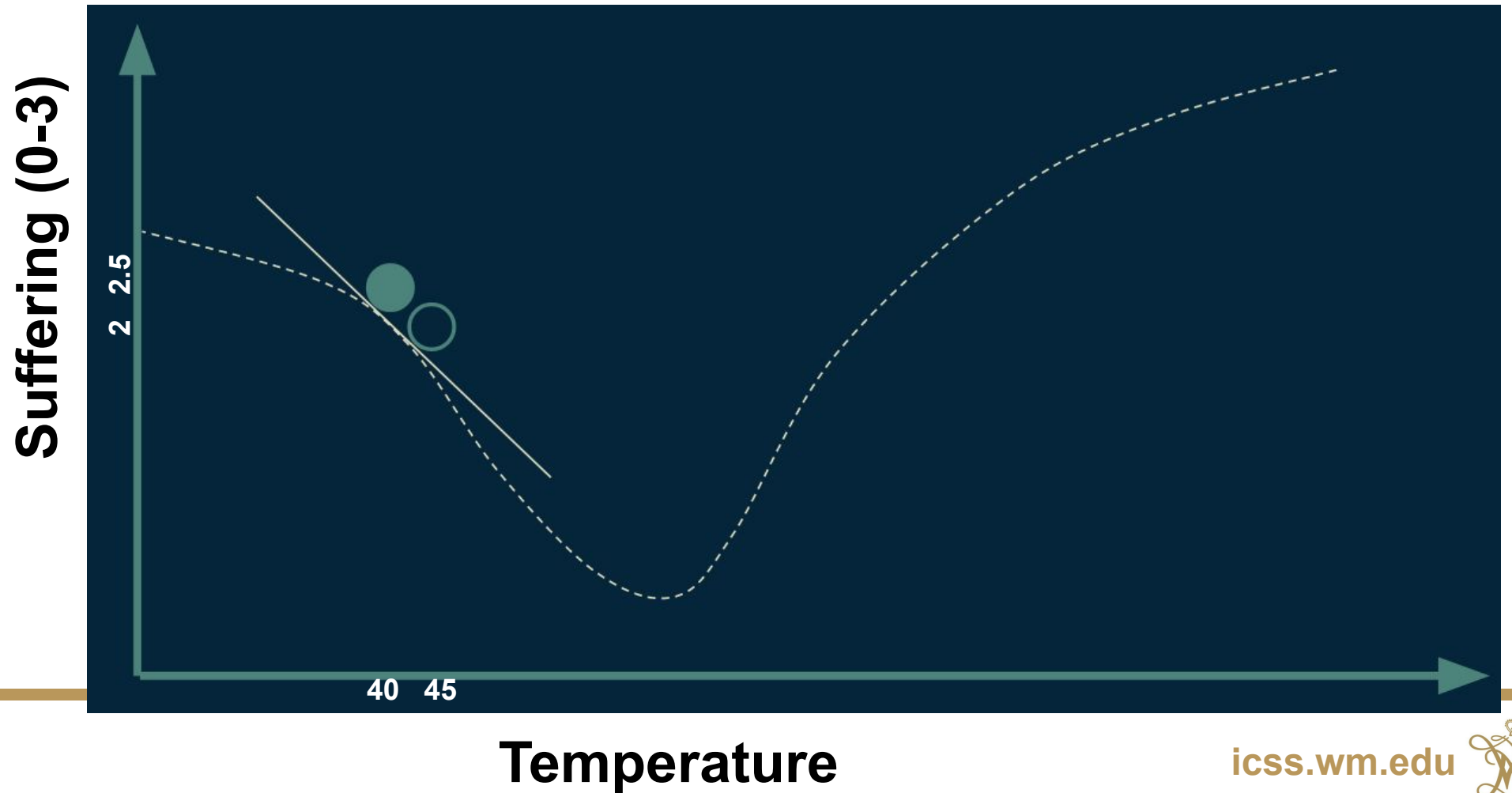
$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

CAOE | CENTER FOR ACCELERATING
OPERATIONAL EFFICIENCY
A DEPARTMENT OF HOMELAND SECURITY CENTER OF EXCELLENCE

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$



$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(40 + 5) - f(40)}{5}$$



$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

In practice we don't just have one variable (temperature). Instead, we have hundreds, thousands, or millions of Weights parameters (**W**). You can imagine calculating a function similar to the one above for each one of those weights parameters, and getting a resultant vector in which you have one slope for every parameter **W**. This vector is called the **gradient**, and the slopes for each **W** are the partial derivatives.

W = [0.34, -1.11, 0.78, 0.12 ... 0.3, 0.77]

Total Loss:
1.25347

Gradient

dW: [?, ?, ?, ? ... ?, ?]

CAOE | CENTER FOR ACCELERATING
OPERATIONAL EFFICIENCY

A DEPARTMENT OF HOMELAND SECURITY CENTER OF EXCELLENCE

W = [0.34, -1.11, 0.78, 0.12 ... 0.3, 0.77] **Total Loss:** 1.25347

h = .0001



W+h: [0.34 + 0.0001, -1.11, 0.78, 0.12 ... 0.3, 0.77]



Gradient

dW: [?, ?, ?, ? ... ?, ?]

W = [0.34, -1.11, 0.78, 0.12 ... 0.3, 0.77] **Total Loss:** 1.25347

h = .0001



W+h: [0.34 + 0.0001, -1.11, 0.78, 0.12 ... 0.3, 0.77]

Total Loss: 1.25322



Gradient

dW: [?, ?, ?, ? ... ?, ?]

W = [0.34, -1.11, 0.78, 0.12 ... 0.3, 0.77] **Total Loss:** 1.25347

W+h: [0.34 + 0.0001, -1.11, 0.78, 0.12 ... 0.3, 0.77]

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Total Loss: 1.25322

$$(1.25322 - 1.25347) / .0001 = -2.5$$

Gradient

dW: [-2.5, ?, ?, ? ... ?, ?]

W = [0.34, -1.11, 0.78, 0.12 ... 0.3, 0.77] **Total Loss:** 1.25347

W+h: [0.34, -1.11+.0001, 0.78, 0.12 ... 0.3, 0.77]

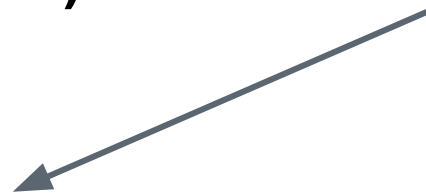
$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Total Loss: 1.25353

$$(1.25353 - 1.25347) / .0001 = 0.6$$

Gradient

dW: [-2.5, 0.6, ?, ? ... ?, ?]



W = [0.34, -1.11, 0.78, 0.12 ... 0.3, 0.77]

W+h: [0.34, -1.11+.0001, 0.78, 0.12 ... 0.3, 0.77]



Gradient

dW: [-2.5, 0.6, 4.3, 0.5 ... 0, 0.3]

Analytic Gradient

$W = [0.34, -1.11, 0.78, 0.12 \dots 0.3, 0.77]$

$$dw = f(X, W)$$
$$\nabla f(X, W) = [\dots]$$

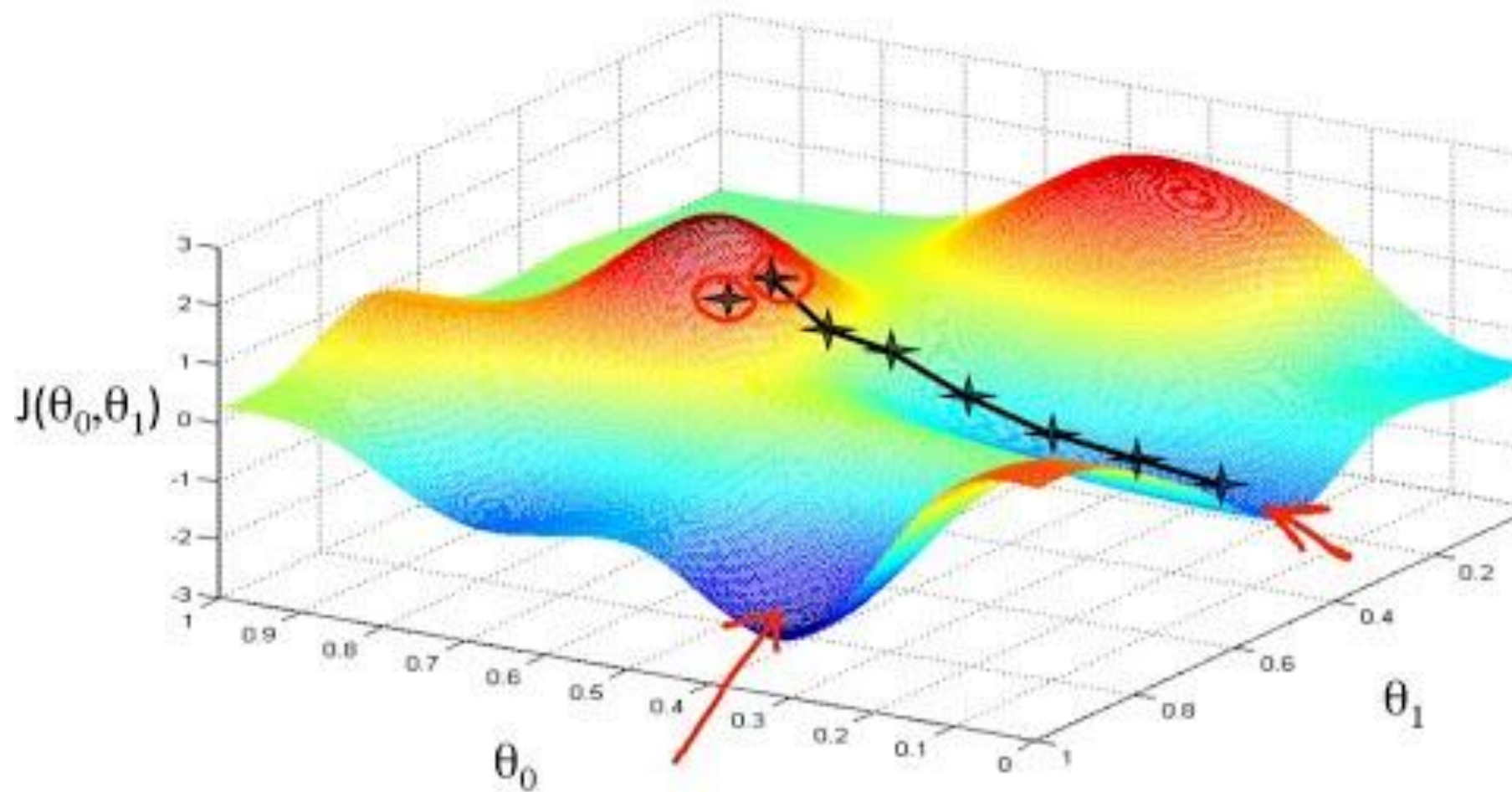
Gradient

$dW: [-2.5, 0.6, 4.3, 0.5 \dots 0, 0.3]$

Gradient Descent in Code

```
maxIterations = 1000
count = 0


while count < maxIterations:
    count = count + 1
    W_gradient_dW = calculateGradient(lossFunction, X, W)
    W = W + -1 * (stepSize * W_gradient_dW)
```

Batch Sizes & Stochastic Gradient Descent

```
maxIterations = 1000
count = 0

while count < maxIterations:
    count = count + 1
    W_gradient_dW = calculateGradient(lossFunction, X, W)
    W = W + -1 * (stepSize * W_gradient_dW)
```



Can be VERY slow for large training datasets.

Batch Sizes & Stochastic Gradient Descent

```
while count < maxIterations:  
    count = count + 1  
    X_sample = X.sample(n=256)  
    W_gradient_dW = calculateGradient(lossFunction, X_sample, W)  
    W = W + -1 * (stepSize * W_gradient_dW)
```

Batch Size (can be anything, normally 8,16,32,64 depending on memory and weights.)

Recap

- What is optimization?
- How does it interrelate with the loss function?
- How can we solve for W using random guessing or an exhaustive search?
- What is gradient descent and stochastic gradient descent, and how does SGD interrelate with batch size?

CAOEF | CENTER FOR ACCELERATING
OPERATIONAL EFFICIENCY
A DEPARTMENT OF HOMELAND SECURITY CENTER OF EXCELLENCE