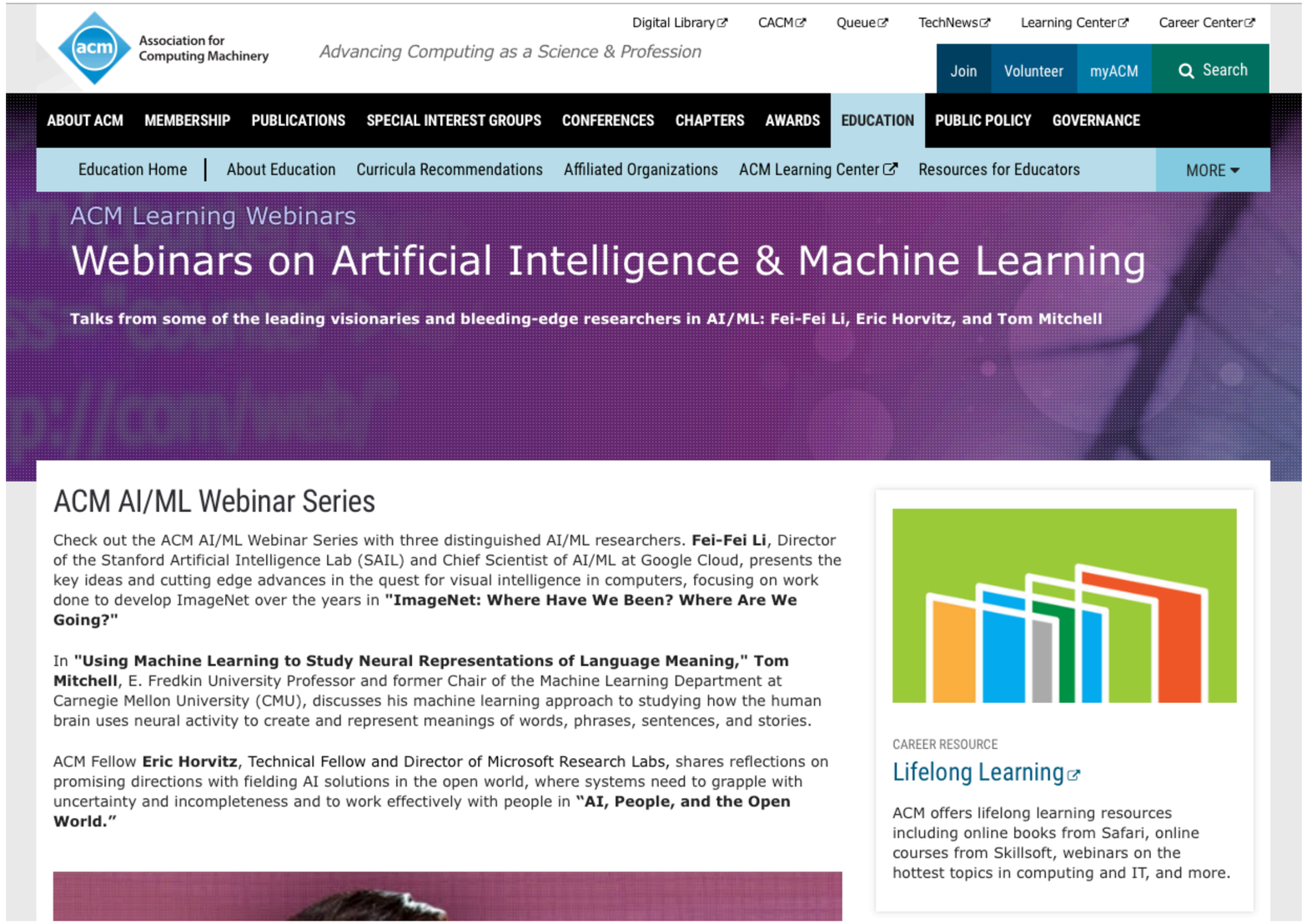
ENGR 4399
Machine Learning

Lesson 12: Deep Learning Concepts

Instructor: Gonzalo D. Parra, PhD

Did You Know?



https://learning.acm.org/webinars

Paper 0:

OpenEDS: Open Eye Dataset

Introduction

- Eye-tracking is critical for VR/AR, autonomous vehicles, and other key research fields.

- Success of eye tracking ML/DL models is in the quality of the data.

    - Large, high-quality datasets are not prevalent, and existing datasets lack labeling for each eye region.

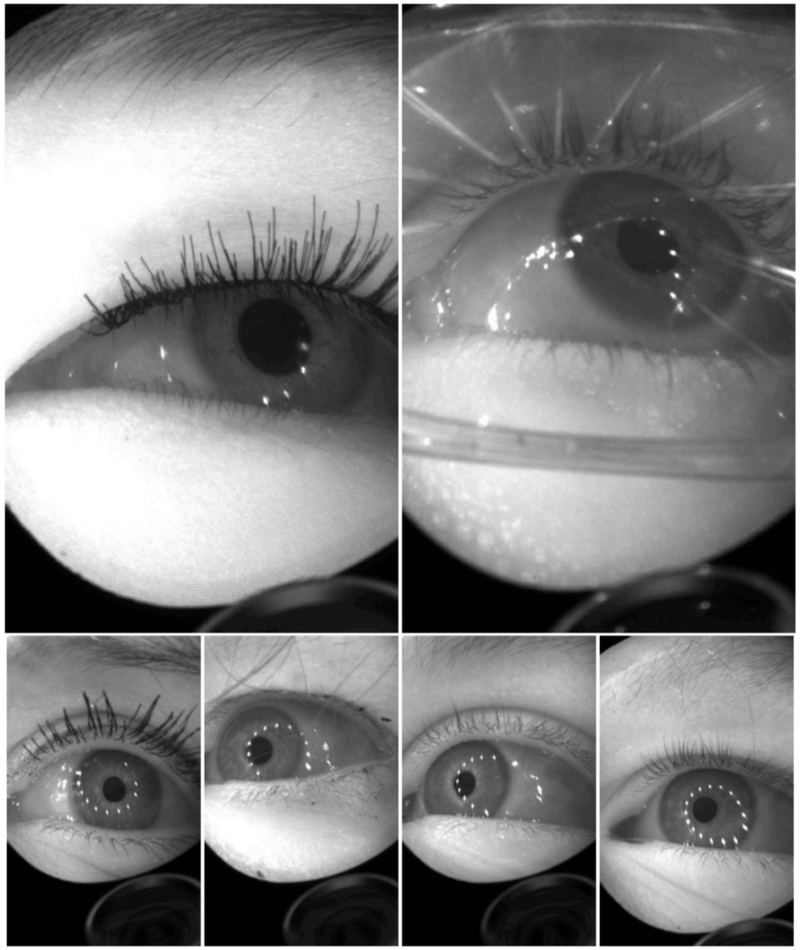
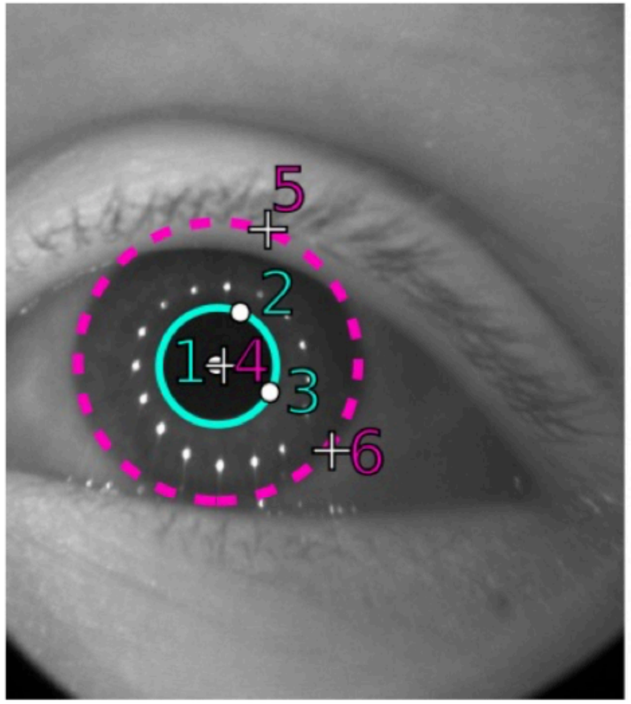


Figure 1: Examples of the acquired HMD images.

Contribution (& Data)



- The **OpenEDS** (Eye Dataset) is proposed in this work.
- There are 356,649 images in controlled environment.

	Sex		Age					Glasses		#Images			#CT.
	Female	Male	18-23	24-30	31-40	41-50	51-65	No	Yes	SeSeg.	IS.	Seq.	
Train	51	44	3	15	39	26	12	92	3	8,916	193,882	57,000	178
Val.	13	15	1	5	5	8	9	27	1	2,403	57,337	16,800	52
Test	18	11	4	3	7	6	9	27	2	1,440	1,471	17,400	56
Total	82	70	8	23	51	40	30	146	6	12,759	252,690	91,200	286

Table 2: Statistics in OpenEDS for train, validation and test. SeSeg.: Images with semantic segmentation annotations. IS.: Images without annotations. Seq.: Image sequence set. CT: corneal topography. We report #identities regarding demographics (sex, age), wearing glasses or not. We also report #images and #corneal topography (represented as point clouds).

Related Works

- Related eye datasets have large number of images, however are either non-uniform, are not in a controlled lighting environment, or are from too few participants.

- These qualities cause problems training and validating DL models.

Dataset	#Images	#Participants	Resolution	FrameRate	Controlled Light	Sync. Left/Right	Optometric Data
STARE [15]	-	-	-	-	No	No	Yes
PoG [28]	-	20	-	30	Yes	No	No
MASD [7]	2,624	82	-	-	No	No	No
Ubiris v2 [32]	11,102	261	400×300	-	No	-	No
LPW [42]	130,856	22	480×640	95Hz	No	No	No
NVGaze [19]	2,500,000	30	480×640	30Hz	Yes	No	No
Gaze Capture [20]	2,500,000	1,450	Various	-	-	No	No
Ours	356,649	152	400×640	200Hz	Yes	Yes	Yes

Table 1: Publicly available datasets in the field of eye tracking. Note that studies in [20] and [32] offer a large number of participants through crowd-sourcing, where images usually include a large portion of the face and hence lack eye details.

BR = Boundary Refinement
SC = Separable Convolution

Metrics

- Metrics used to evaluate the efficacy of the dataset were extensive. *
- In further evaluation, deep learning models (derived from SegNet) were leveraged to predict eye regions in each image.

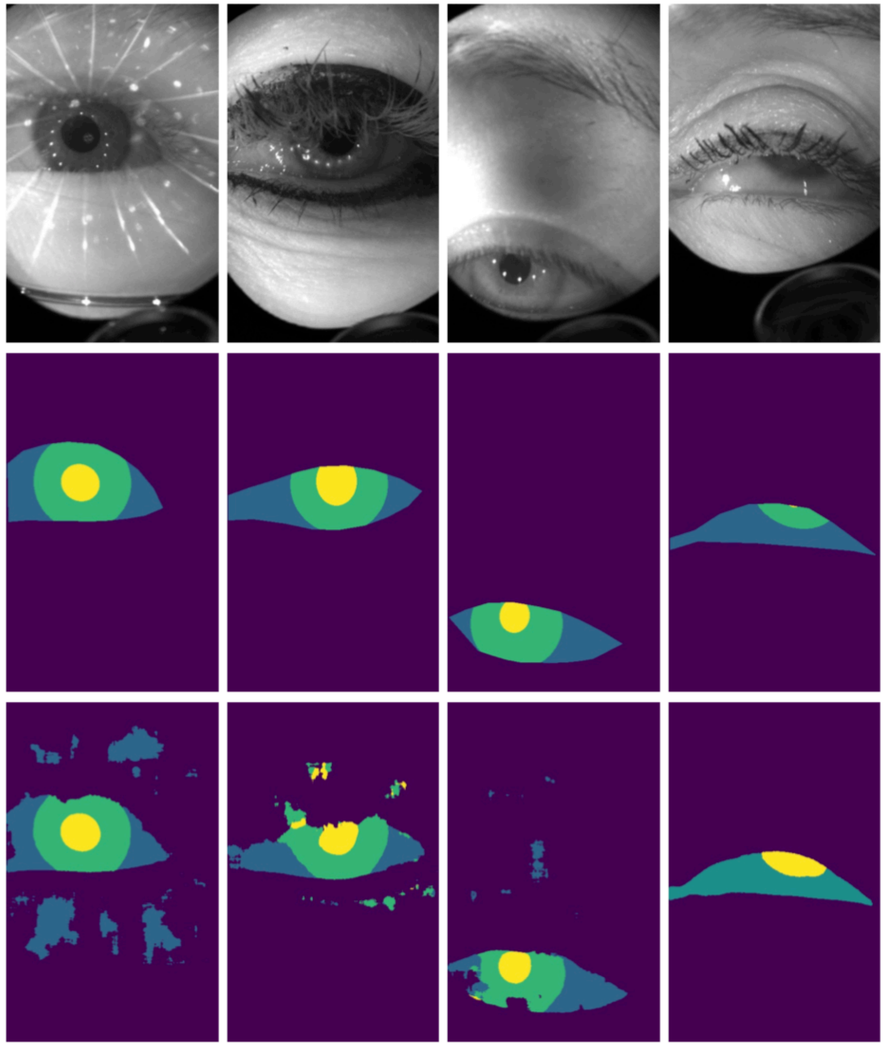
Model	Pixel acc.	Mean acc.	Mean F1	Mean IoU	Model Size (MB)	#Param. (M)
mSegnet	98.0	96.8	97.9	90.7	13.3	3.5
mSegnet w/ BR	98.3	97.5	98.3	91.4	13.3	3.5
mSegnet w/ SC	97.6	96.6	97.4	89.5	1.6	0.4

Table 4: Results on semantic segmentation. Model size: size of the PyTorch models on disk. #Params: the number of learnable parameters, where "M" stands for million. *mSegnet*: 4-layer segnet with multiplicative skip connection between the last layer of the encoder and the decoder network

**this explanation should have been better!*

Results

- Results of “challenging samples”:
  - Top: Original image
  - Center: Ground truth
  - Bottom: Predictions from SegNetBR



Eyeglasses | Mascara | Dim light | Varying pupil size

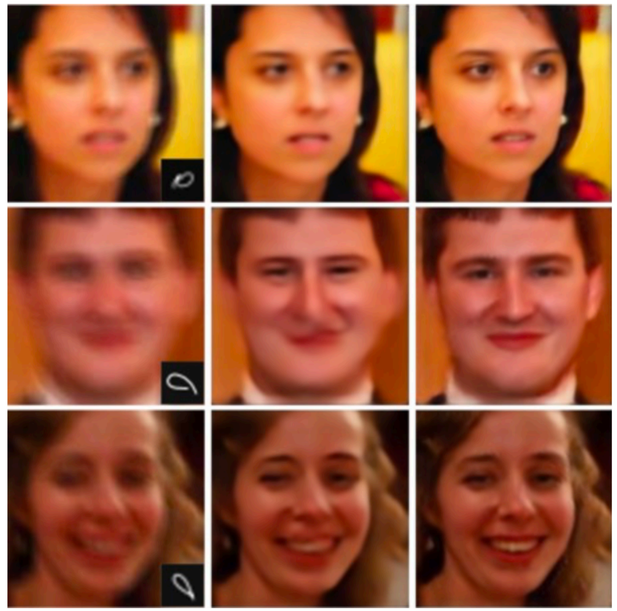
Research Project

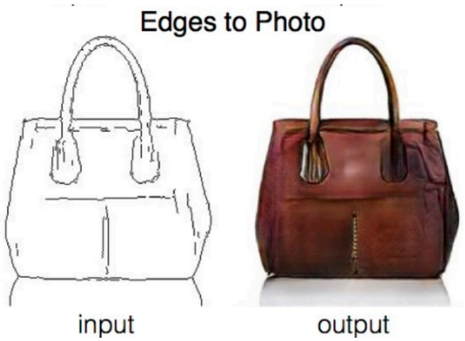
Ideas..

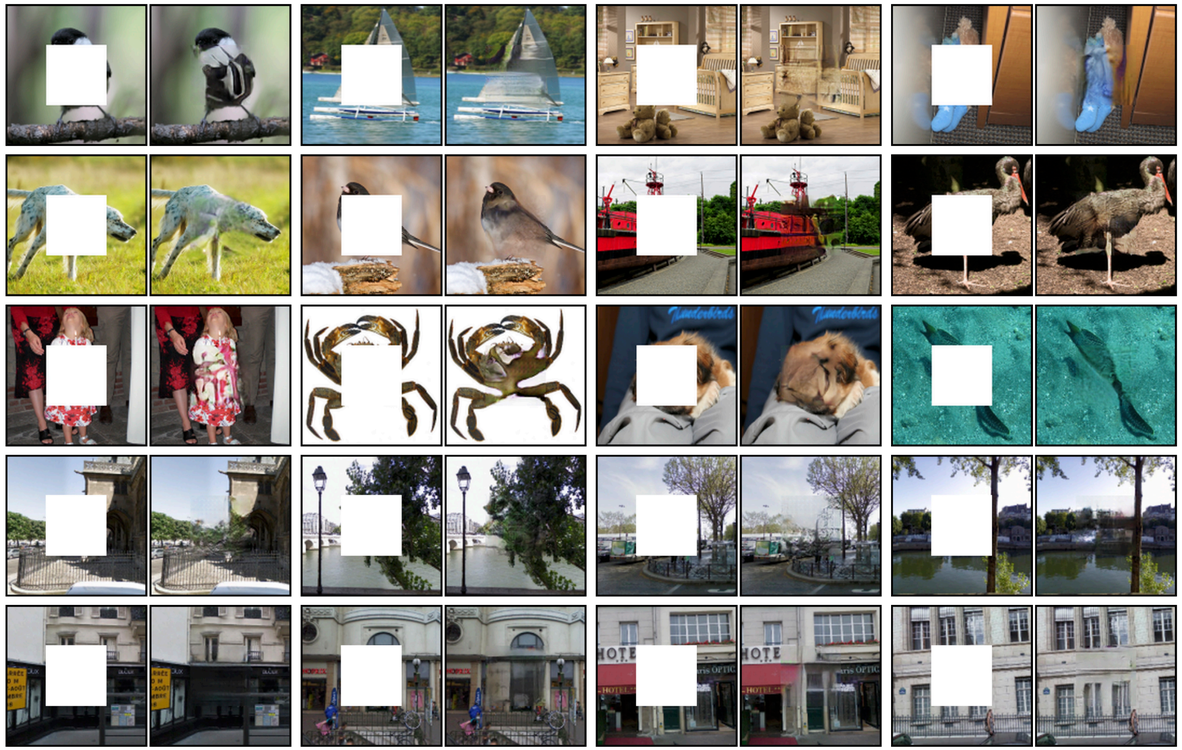
- Some open problems in DL:
    - Understanding How Neural Networks Think
    - Progressive Neural Networks
- Applications:
    - Automatic image/video colorization
    - Pose estimation, behavior estimation
    - Image/video annotation
    - Generating photos of [faces, galaxies, you name it!]

(b) Handbag images (input) & **Generated** shoe images (output)

Deep Semantic Face Deblurring







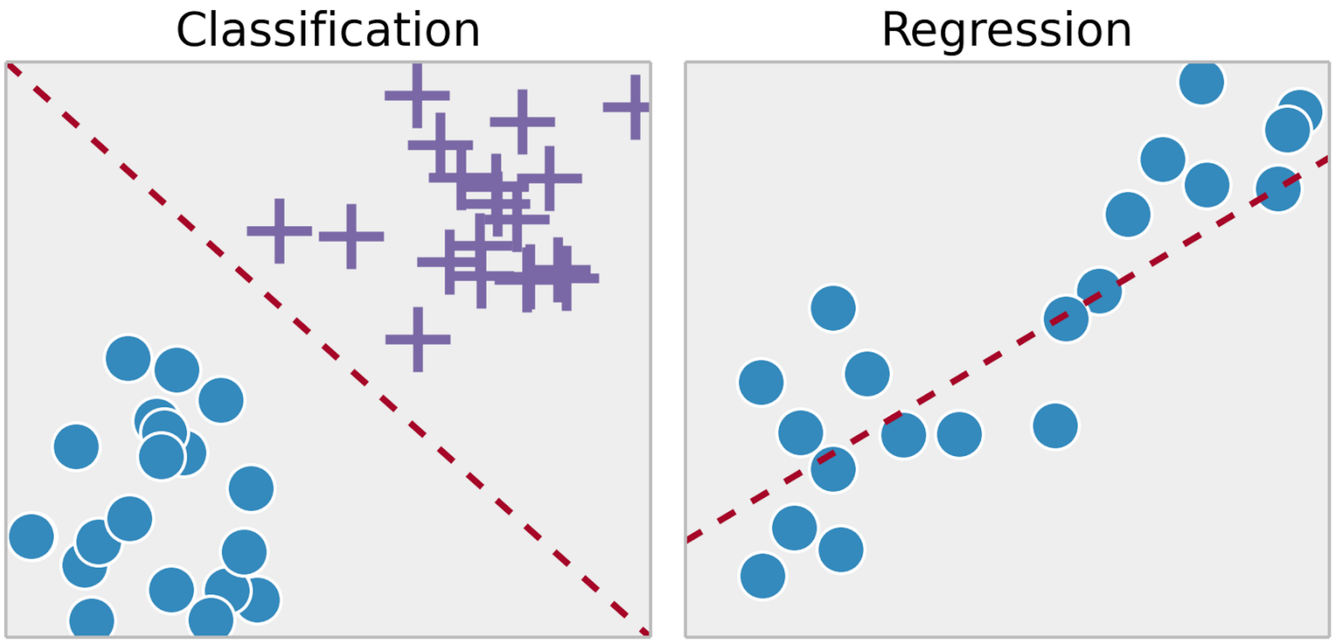
[Synthesizing Audio with Generative Adversarial Networks]

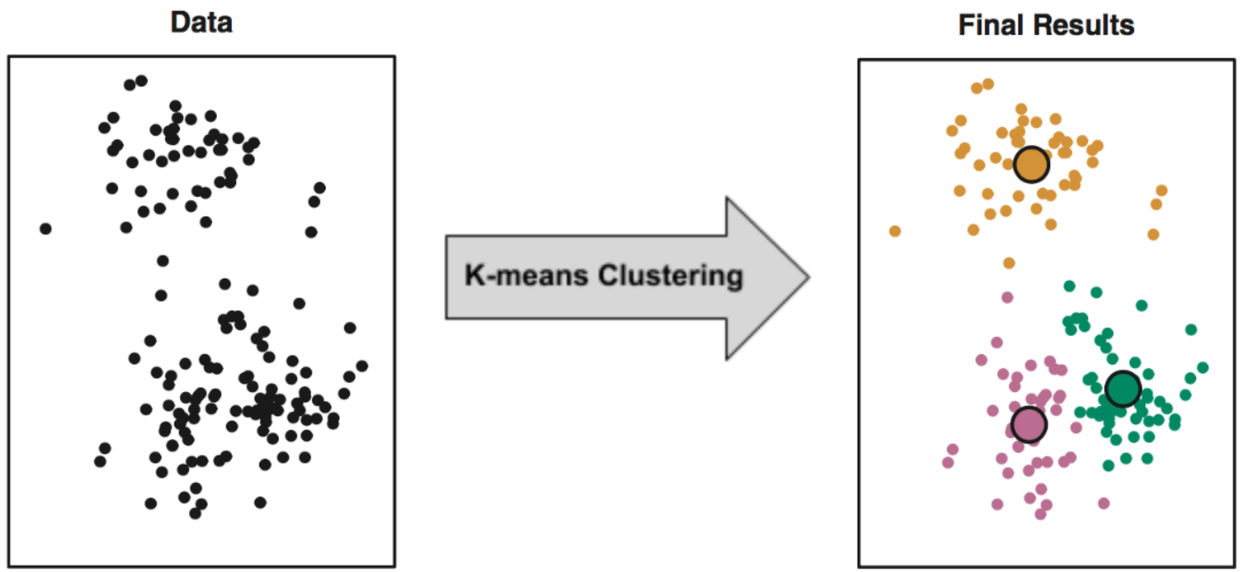
Machine Learning

Supervised Learning

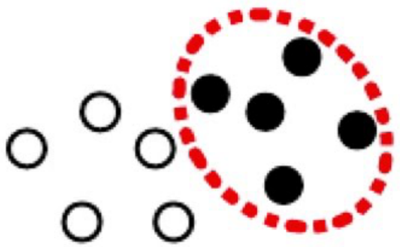
&

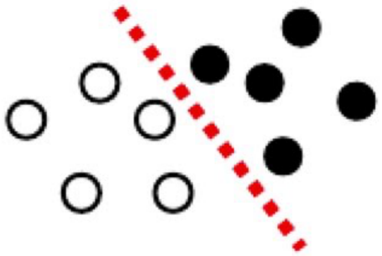
Unsupervised Learning





Generative vs. Discriminative Models



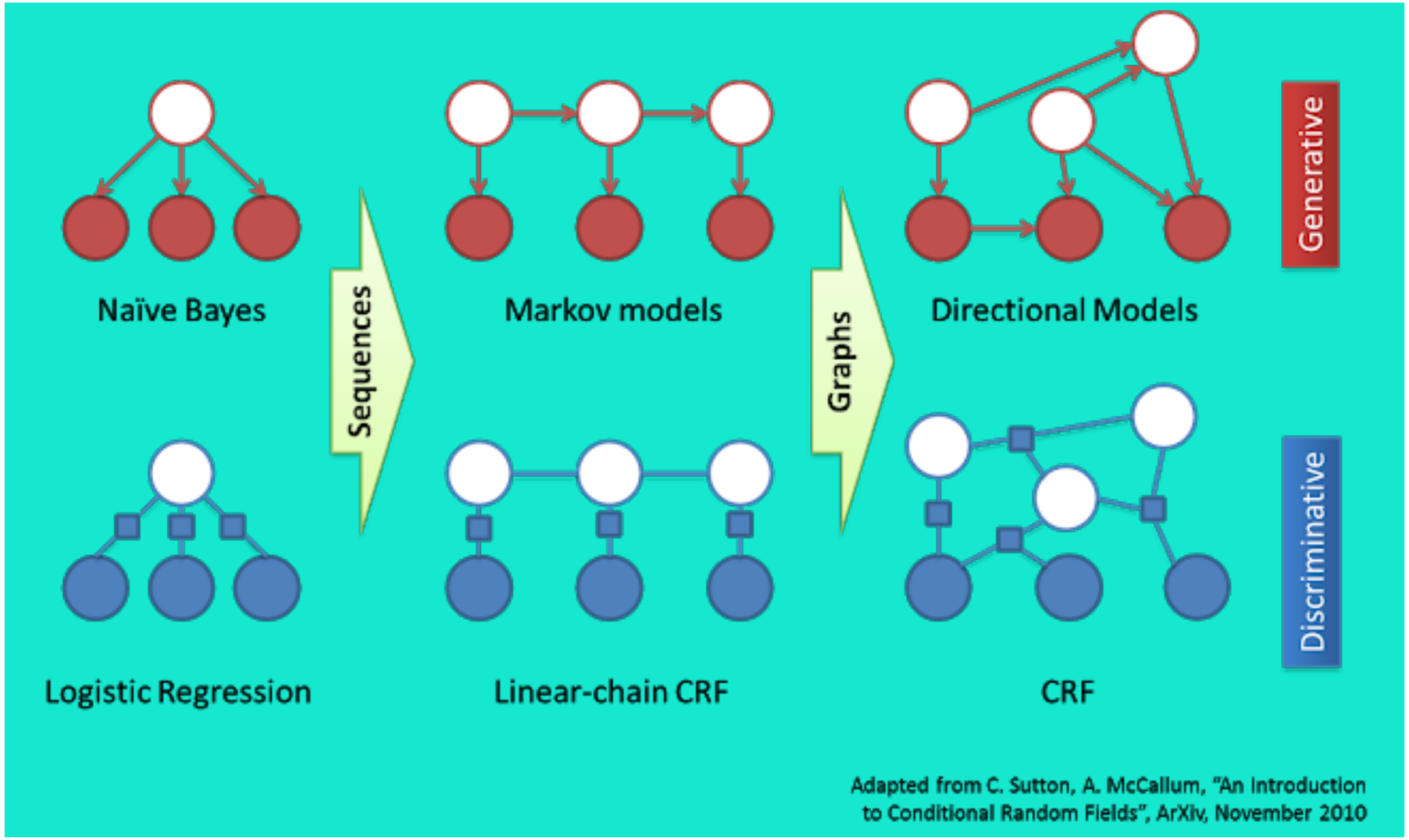


Probabilistic model of class(es)	**Goal**	Decision boundary
Unlabeled data*	**Data**	Labeled data
Where one model becomes more likely	**Focus**	Boundary between classes
Naive Bayes Classifiers Gaussian Models	**Examples**	Perceptrons SVMs Nearest Neighbor

Source: *Linear Regression Analysis: Theory and Computing*, Xin Yan

Generative vs. Discriminative Models

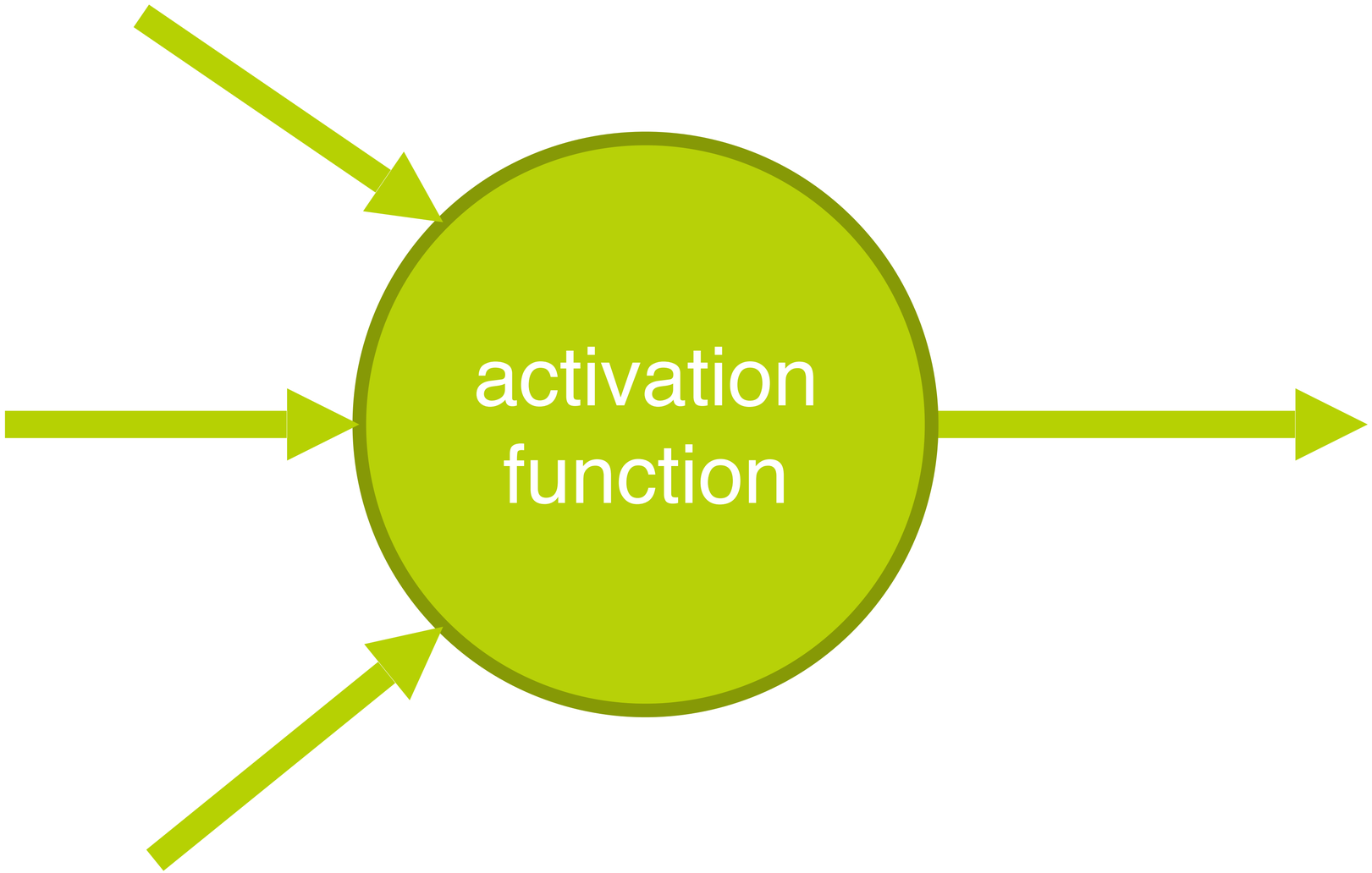
Looking ahead..



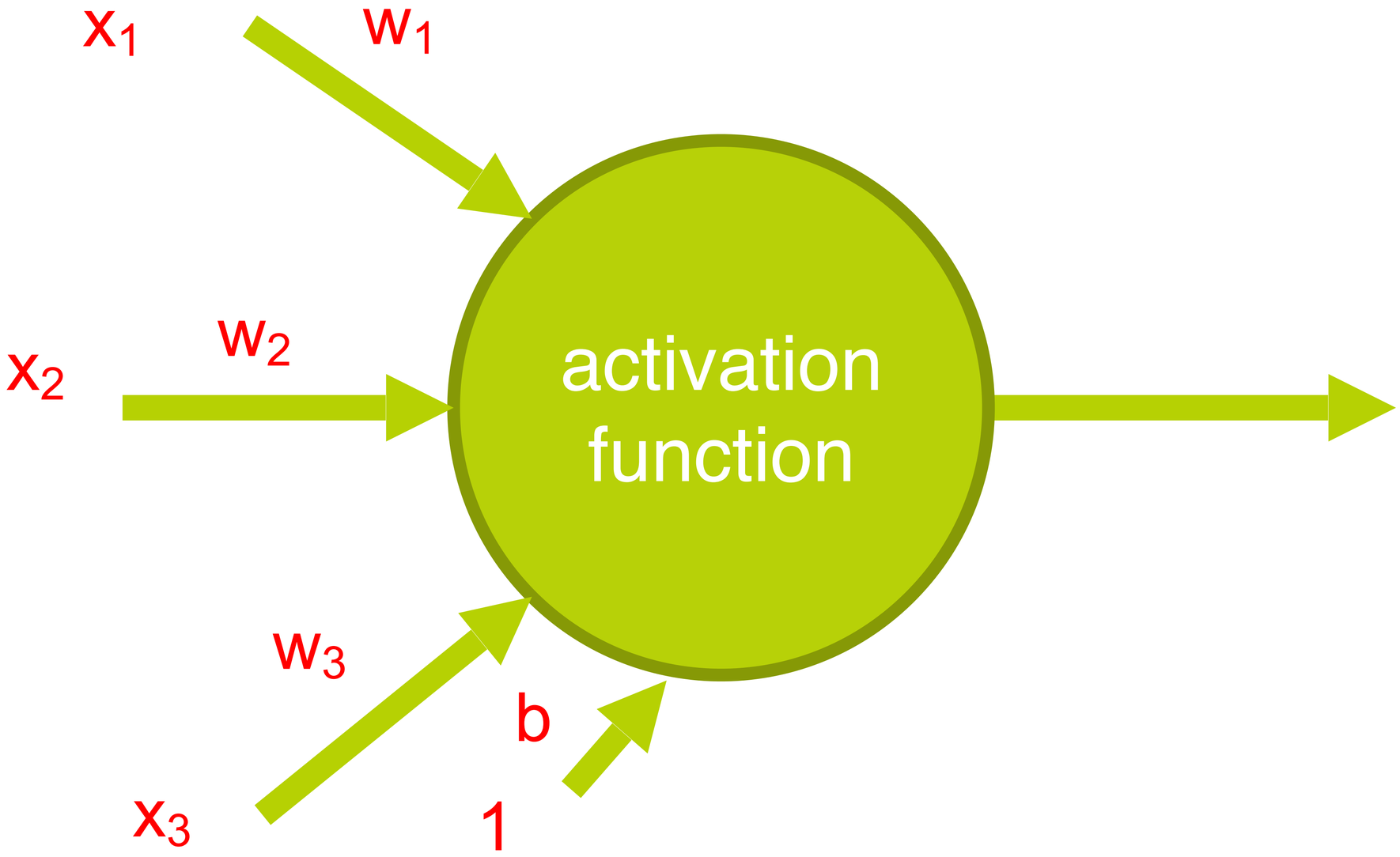
Source: *Linear Regression Analysis: Theory and Computing*, Xin Yan

Neural Networks

Basic Neuron Visualization

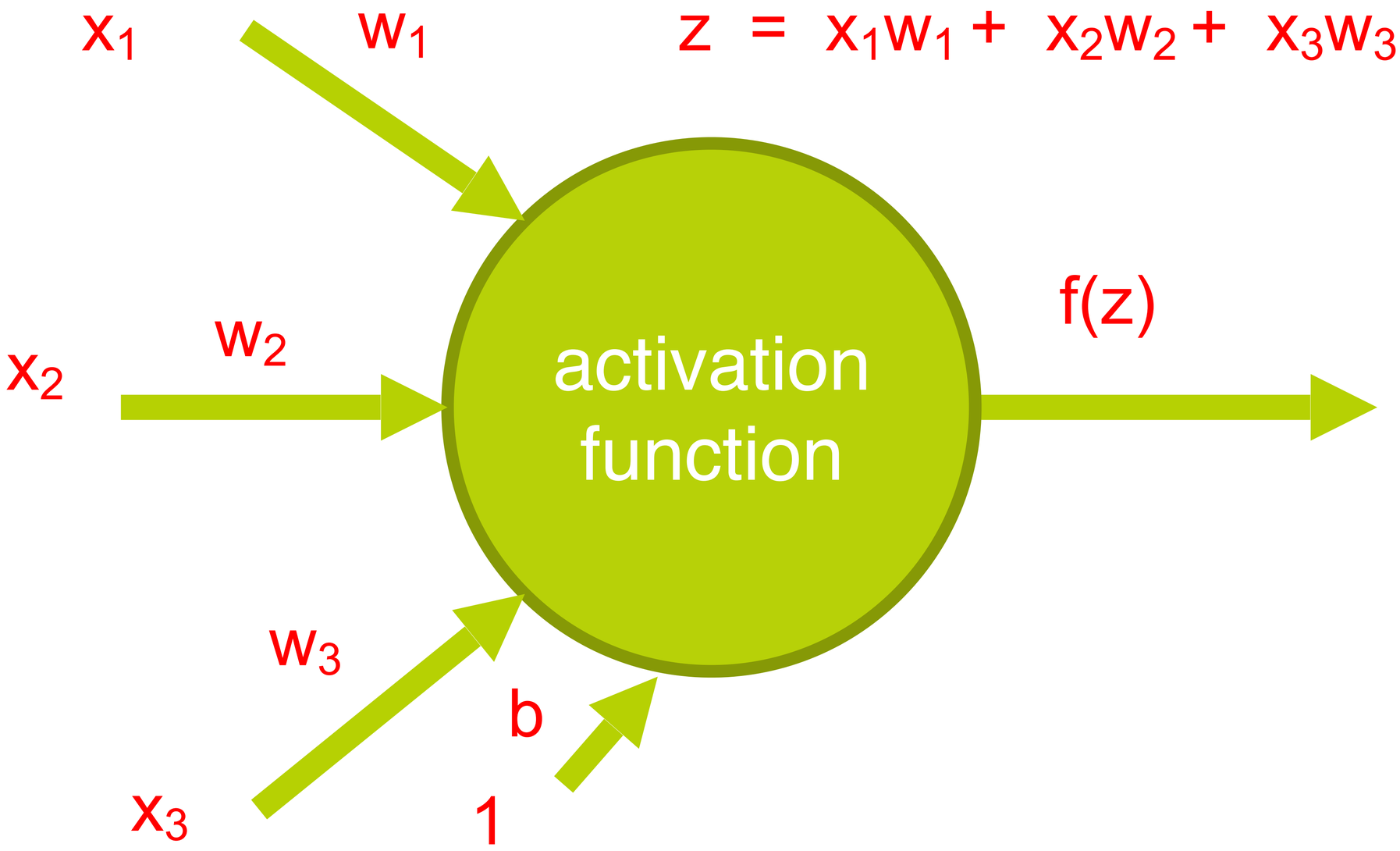


Basic Neuron Visualization

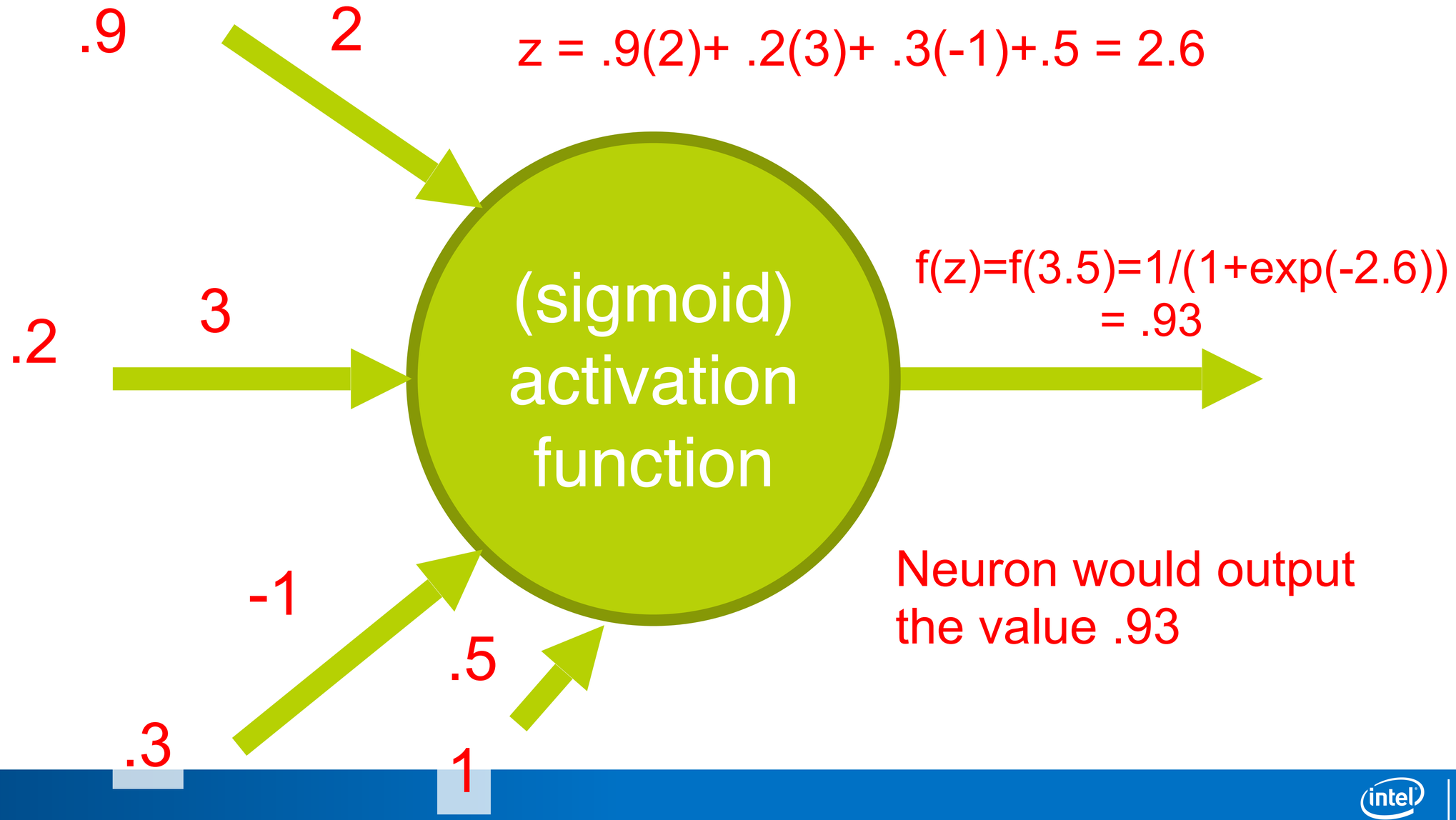


Basic Neuron Visualization

$$z = x_1w_1 + x_2w_2 + x_3w_3 + b$$



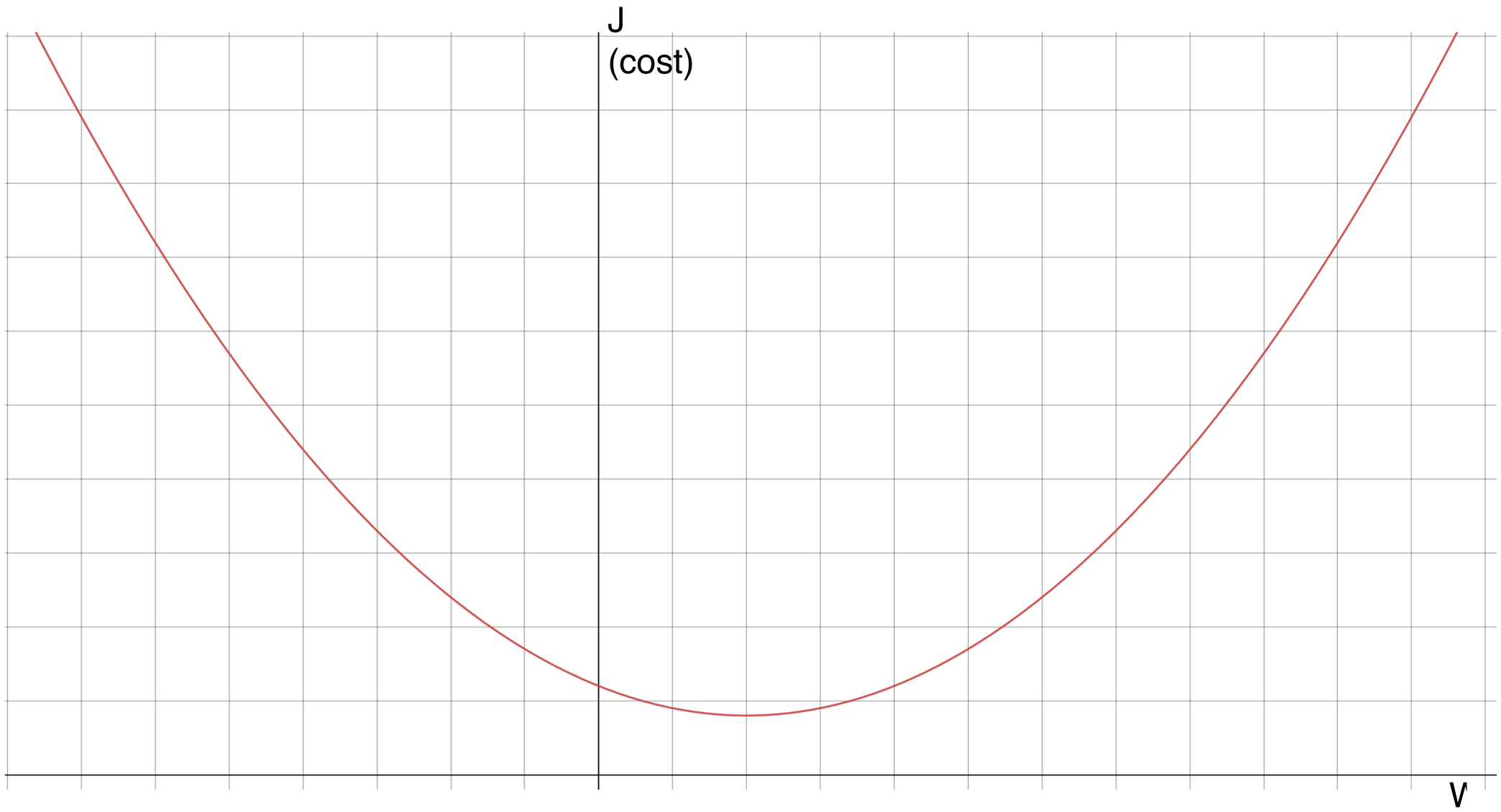
Example Neuron Computation



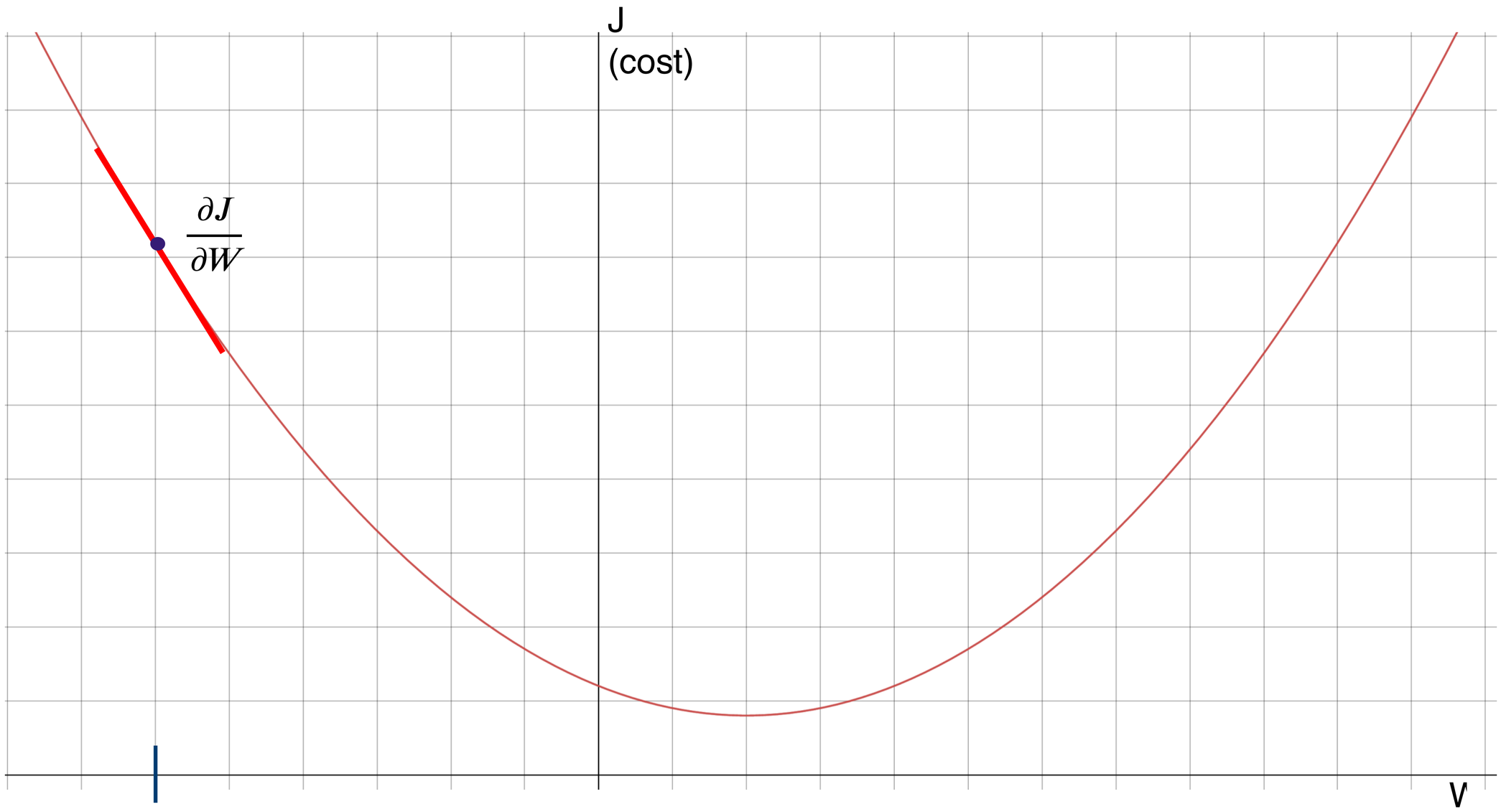
Activity:

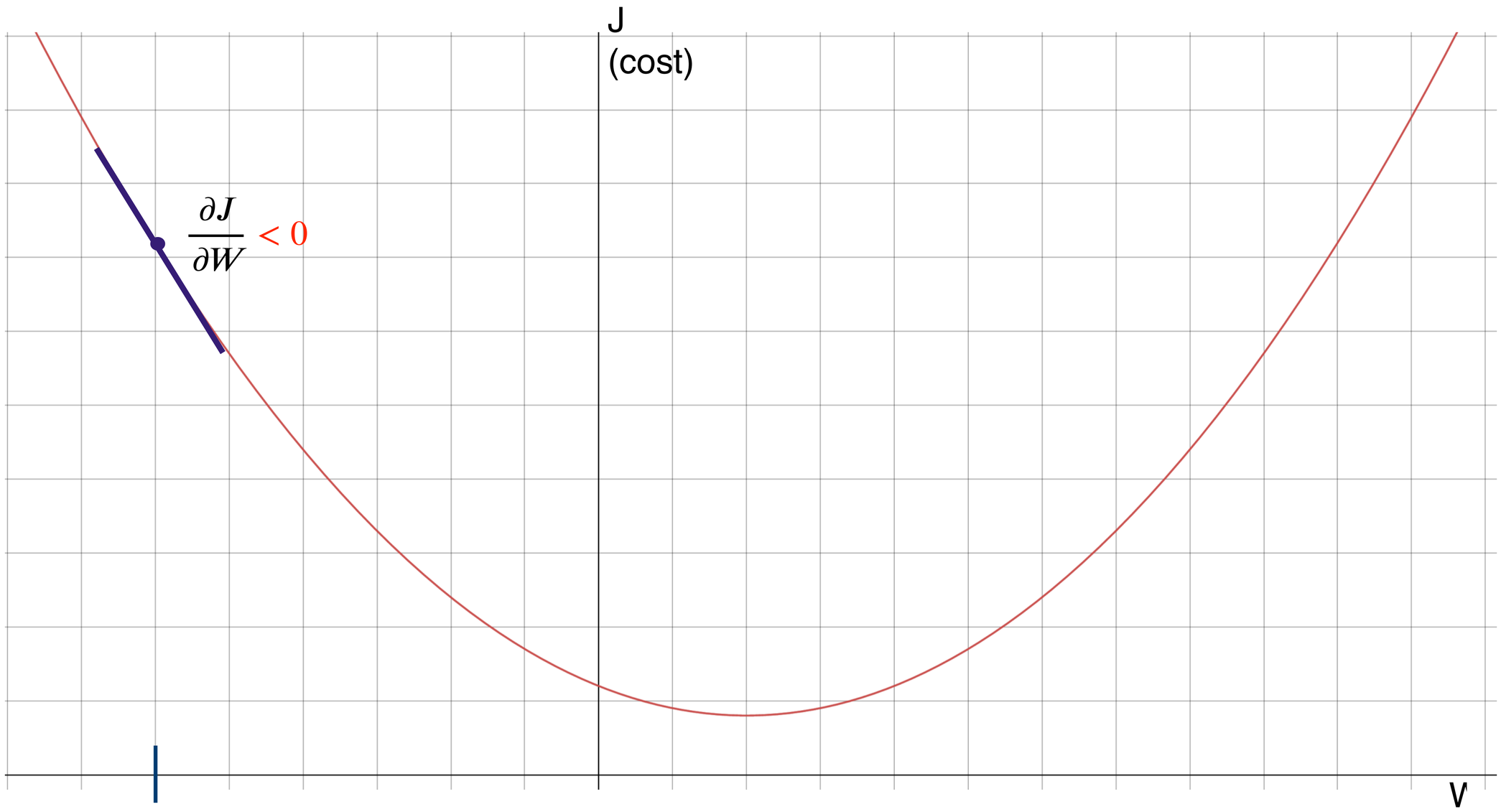
How Does Gradient Descent *Work*?

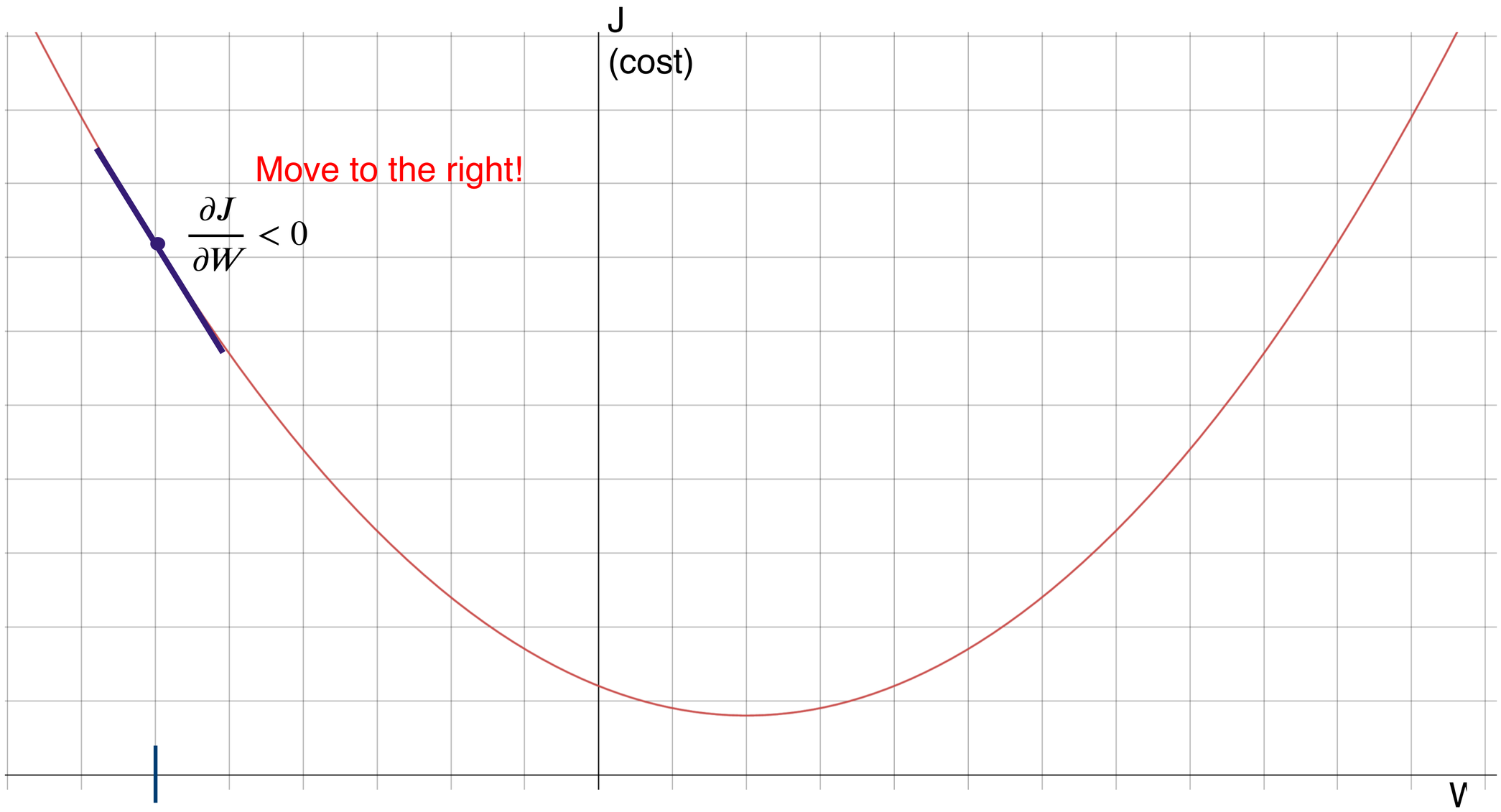
Gradient Descent

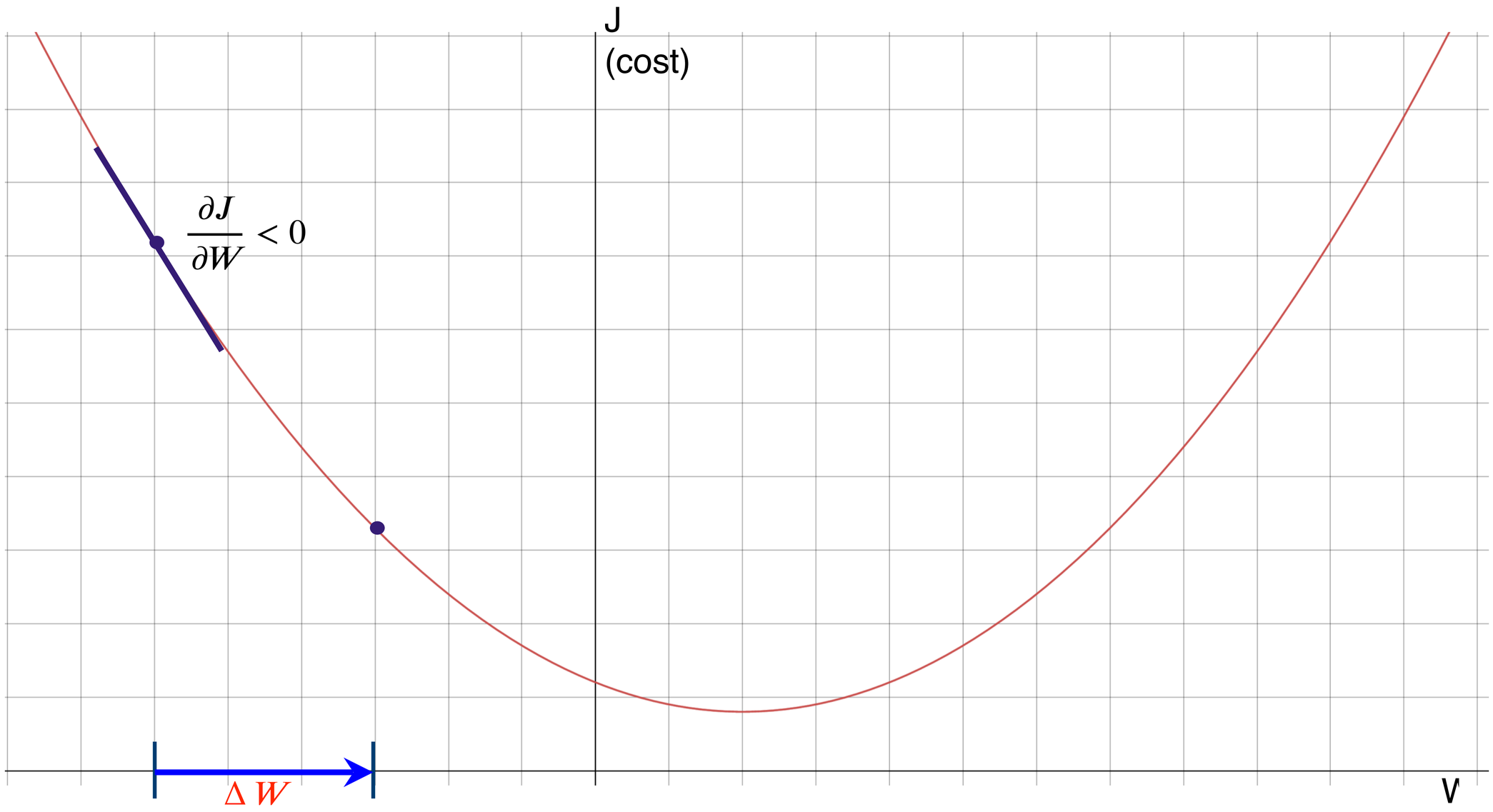

J
(cost)
V

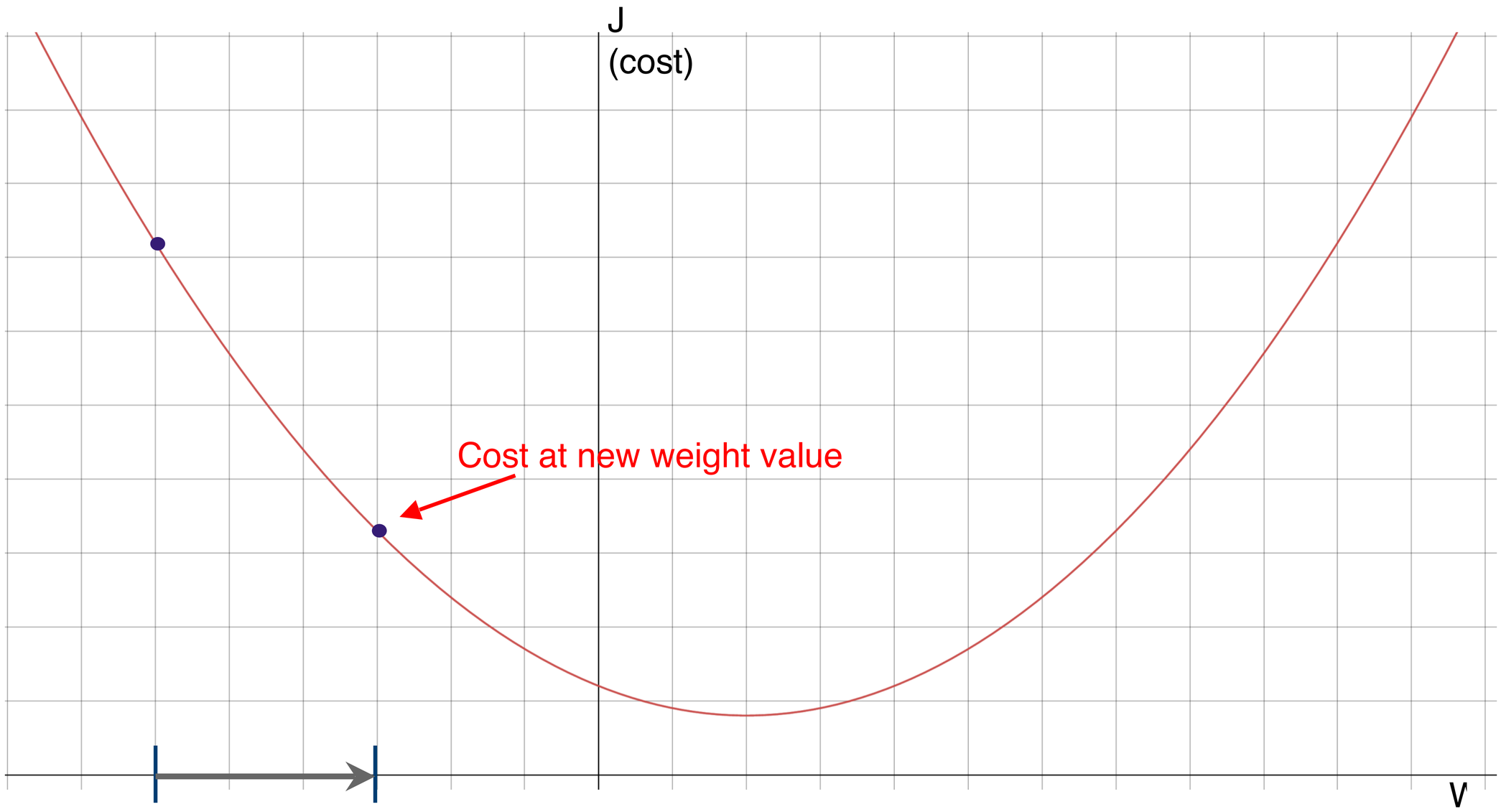
J
(cost)
Random guess put us here
V

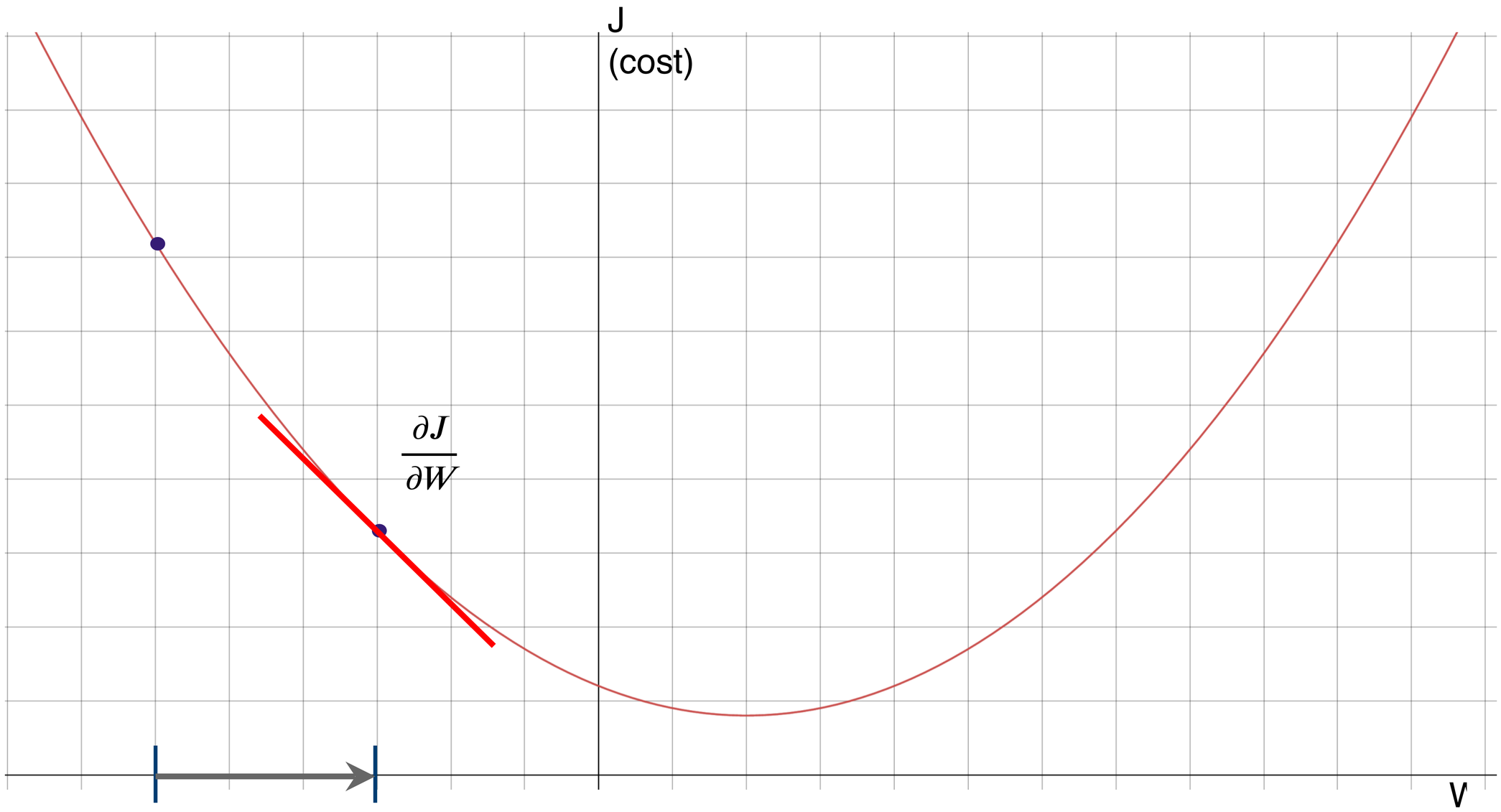

J
(cost)
$\frac{\partial J}{\partial W}$
W

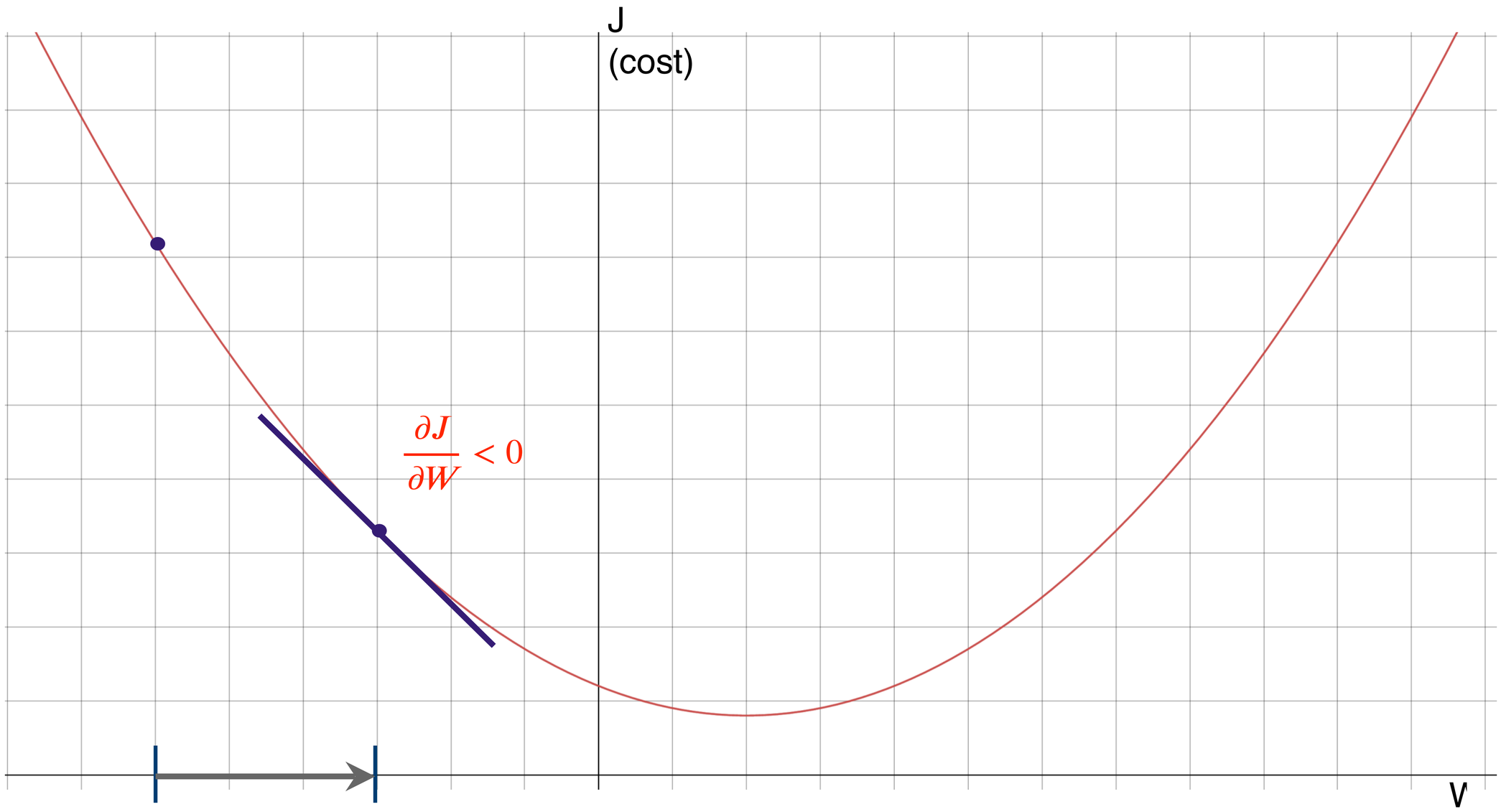

J
(cost)
$\frac{\partial J}{\partial W} < 0$
W

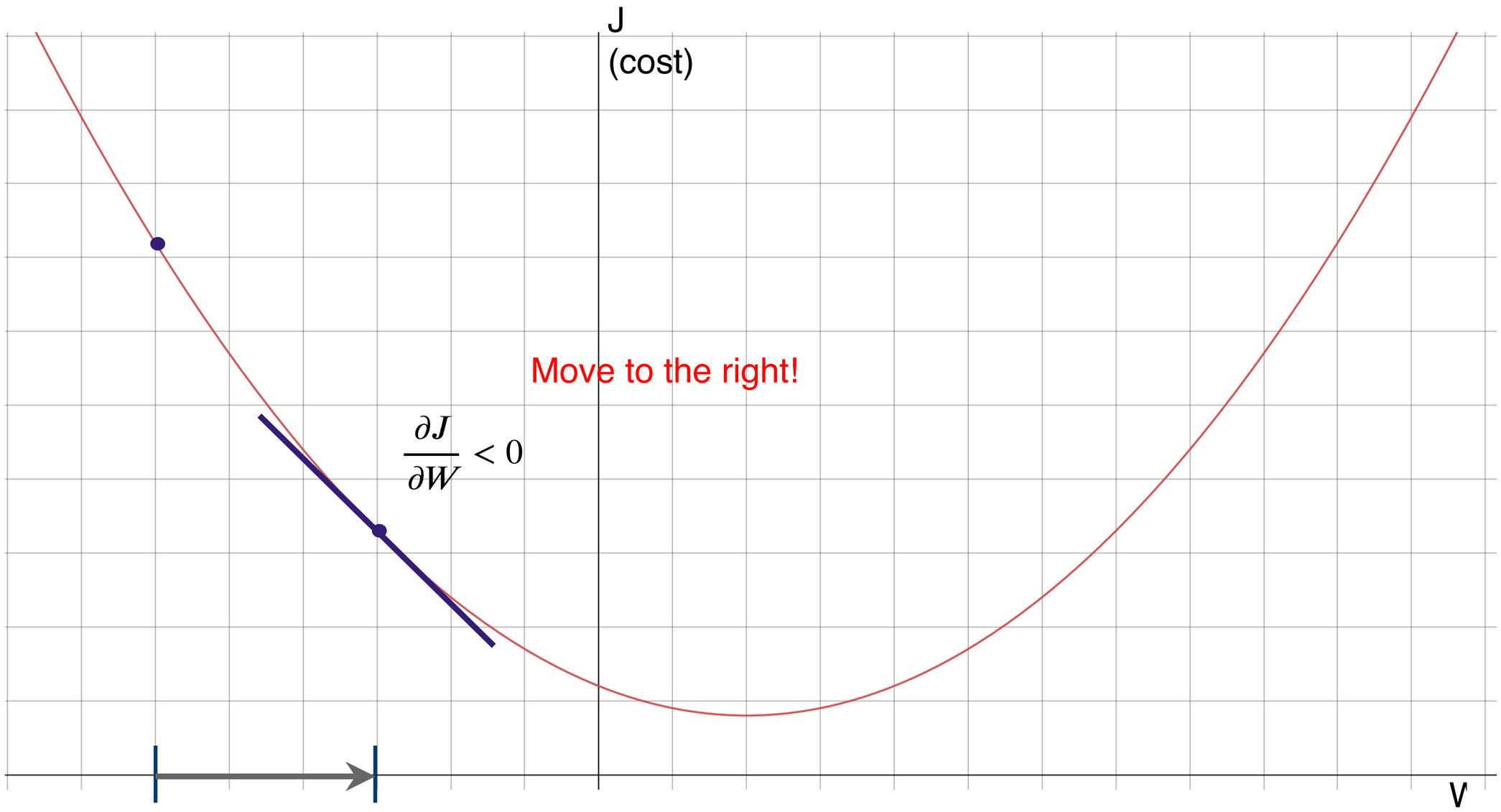

J
(cost)
Move to the right!
$\frac{\partial J}{\partial W} < 0$
W

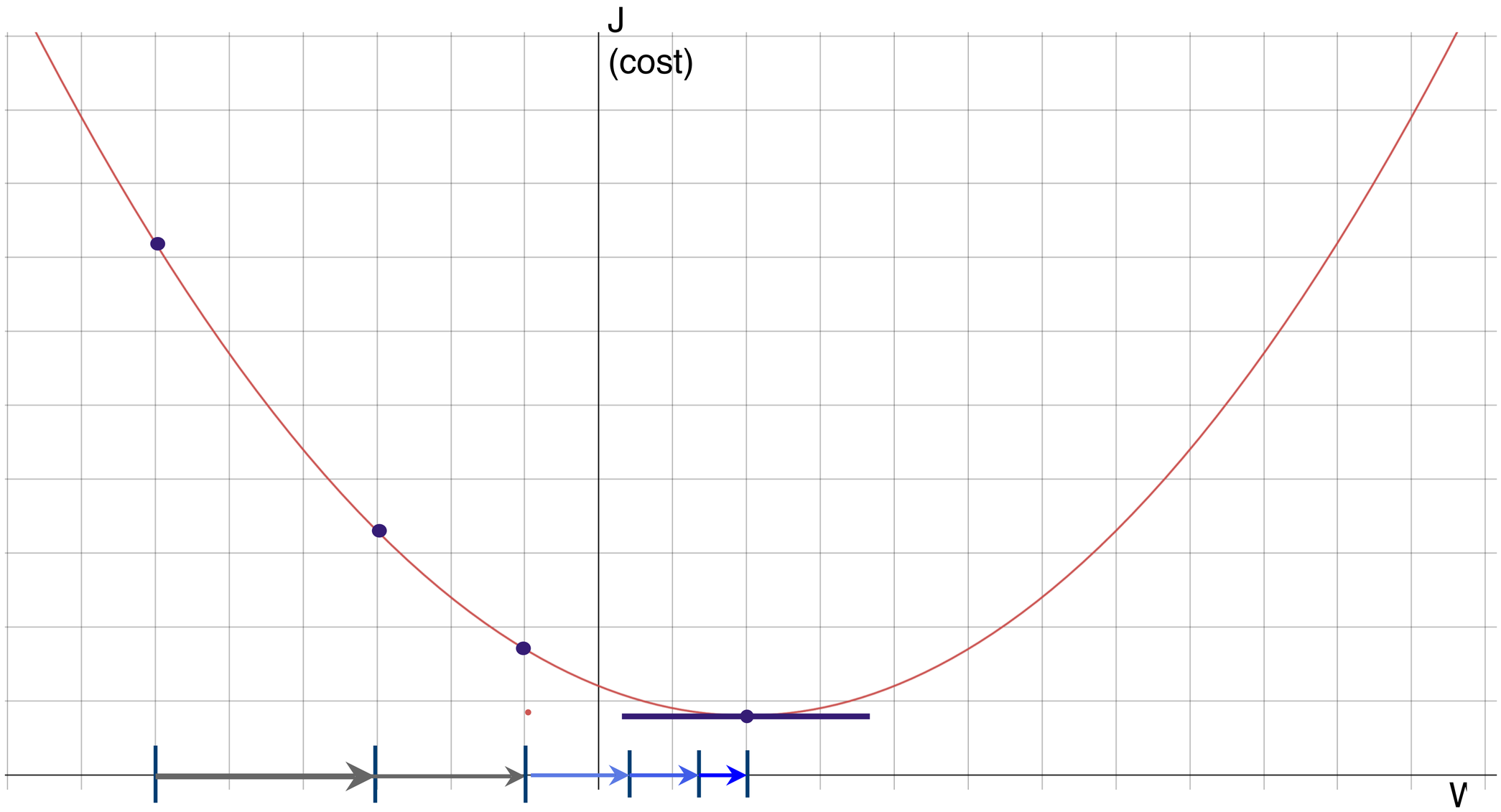

J
(cost)
$\frac{\partial J}{\partial W} < 0$
ΔW
W


J
(cost)
Cost at new weight value
V


J
(cost)
$\frac{\partial J}{\partial W}$
W


J
(cost)
$\frac{\partial J}{\partial W} < 0$
W


J
(cost)
Move to the right!
$\frac{\partial J}{\partial W} < 0$
W


J
(cost)
V

The process of doing gradient descent (math-version)

1. Find the derivative of loss w.r.t to weights over training data

- Plug data into our derivative function, and sum up over data points

The number we'll use to adjust the weight

$$\Delta W = \sum_{i=1}^{n} \frac{\partial J}{\partial W}(x_i,\ y_i)$$

$$\frac{\partial J}{\partial W}(x_i, y_i) = \frac{1}{n}\sum_{i=1}^{n} x_i\left(\hat{y}_i - y_i\right)$$

Derivative of MSE

The process of doing gradient descent (math-version)

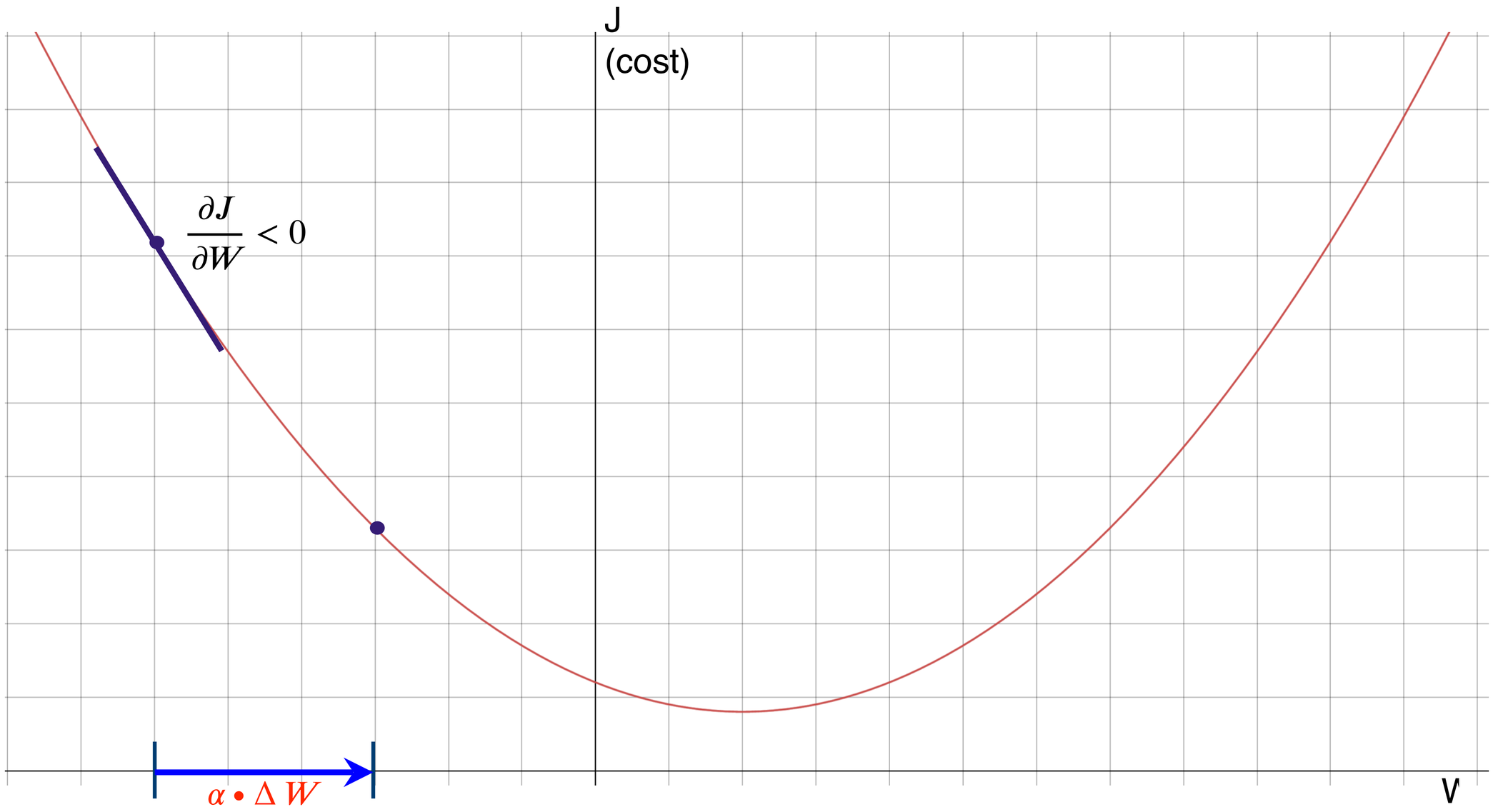
2. Adjust the weight by subtracting some amount of
   - (alpha) is known as the *learning rate*
   - It's the first *hyper-parameter* we've seen in the class

$$W := W - \alpha \bullet \Delta W$$

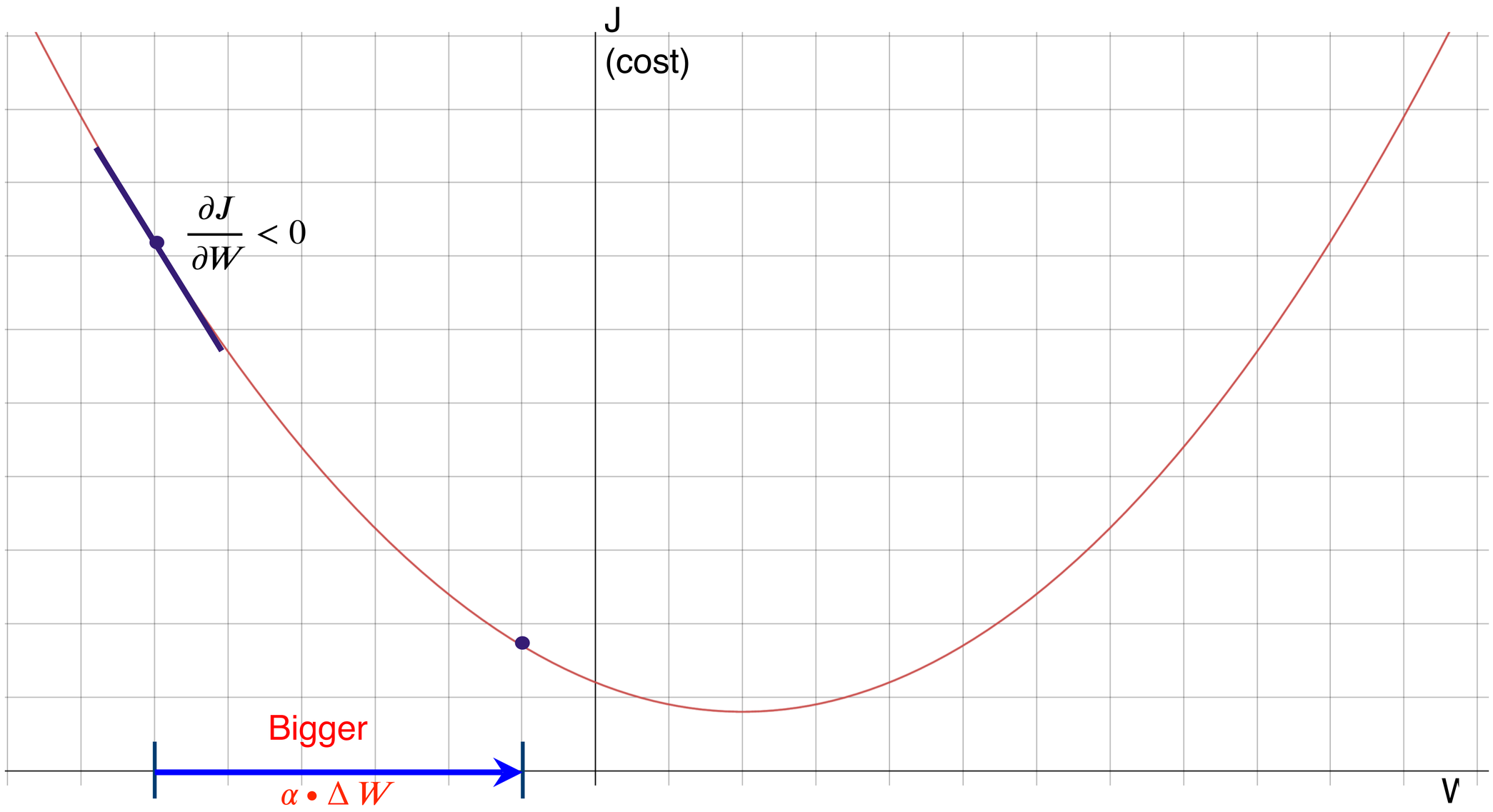
Minus adjusts W in the correct direction

3. Repeat until model is "done training"
   – We can also adjust the learning rate as we train

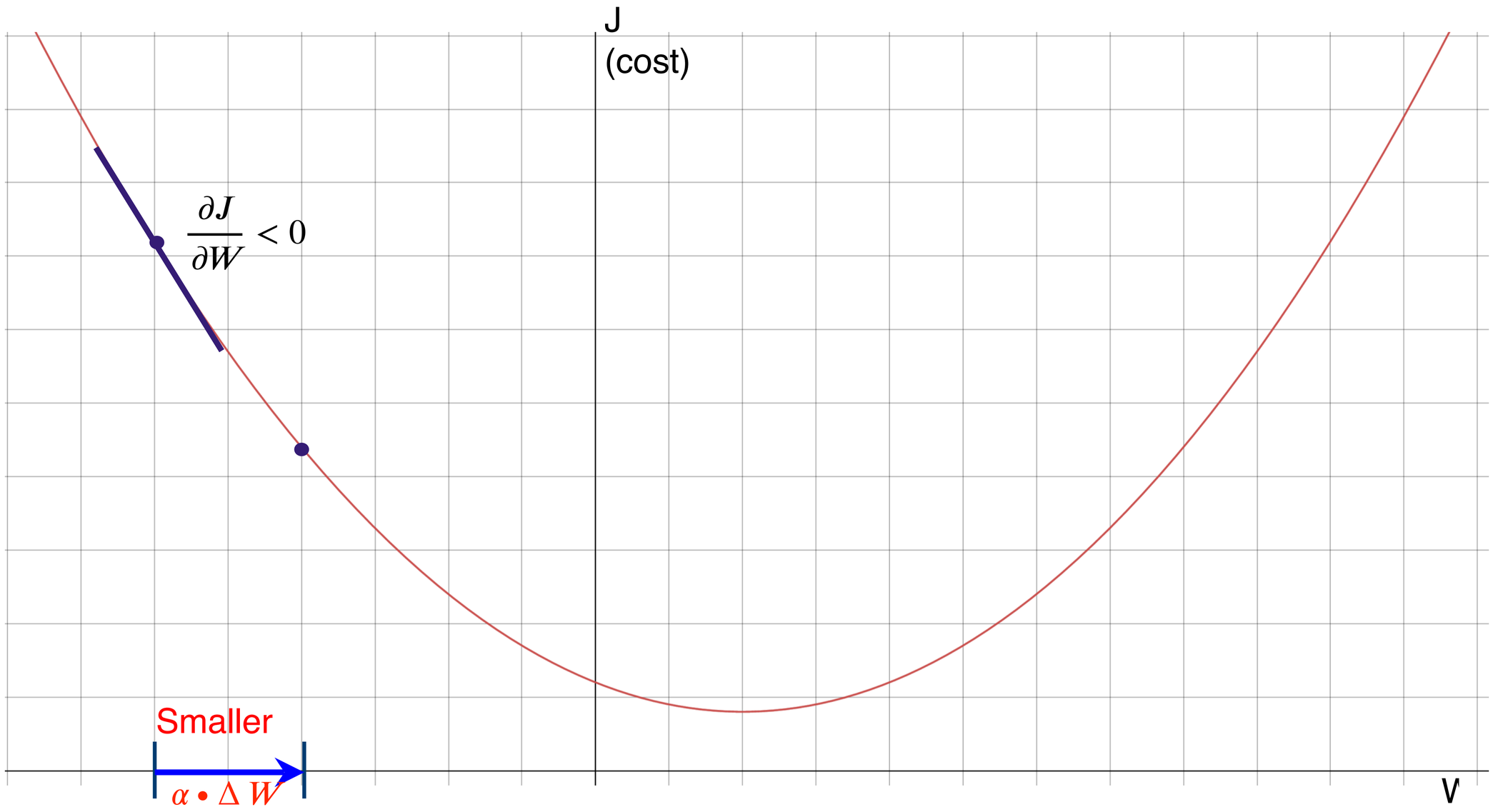
Adjusting the learning rate



Adjusting the learning rate

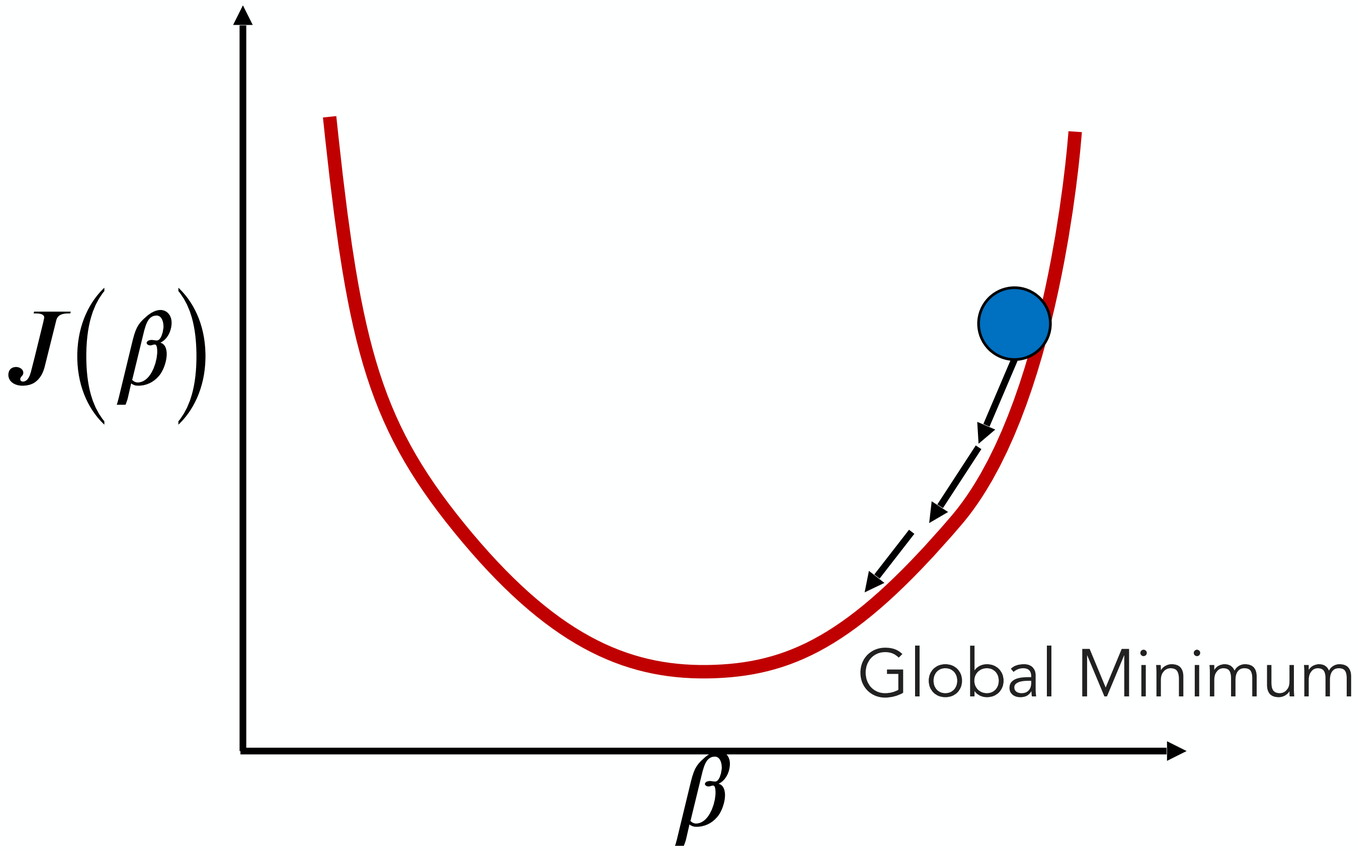


Adjusting the learning rate



Gradient Descent

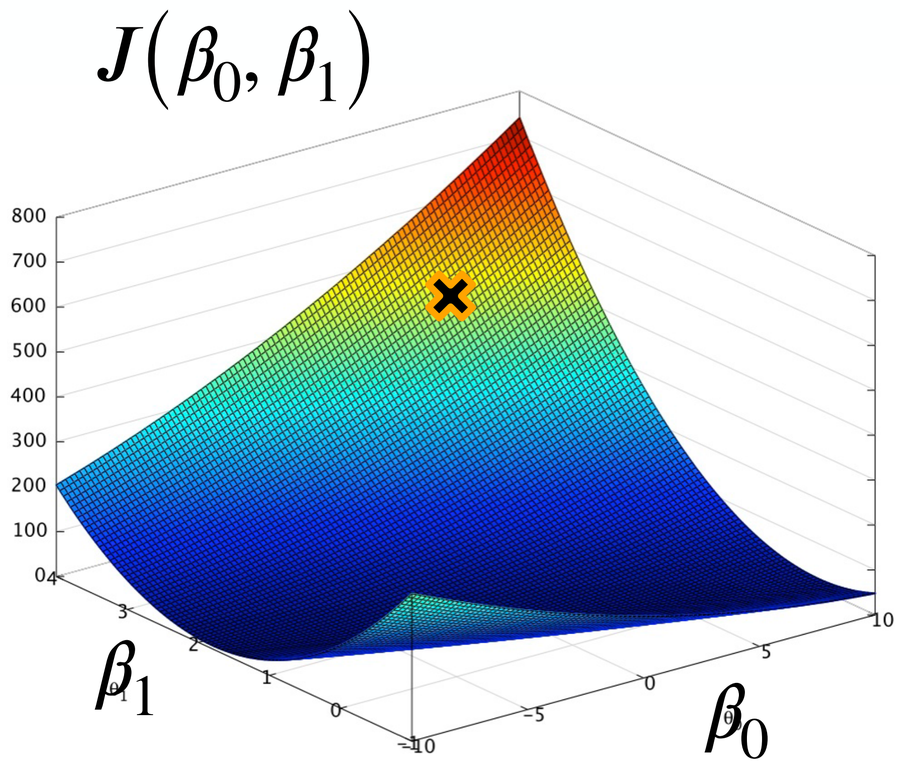
Start with a cost function J():



Then gradually move towards the minimum.

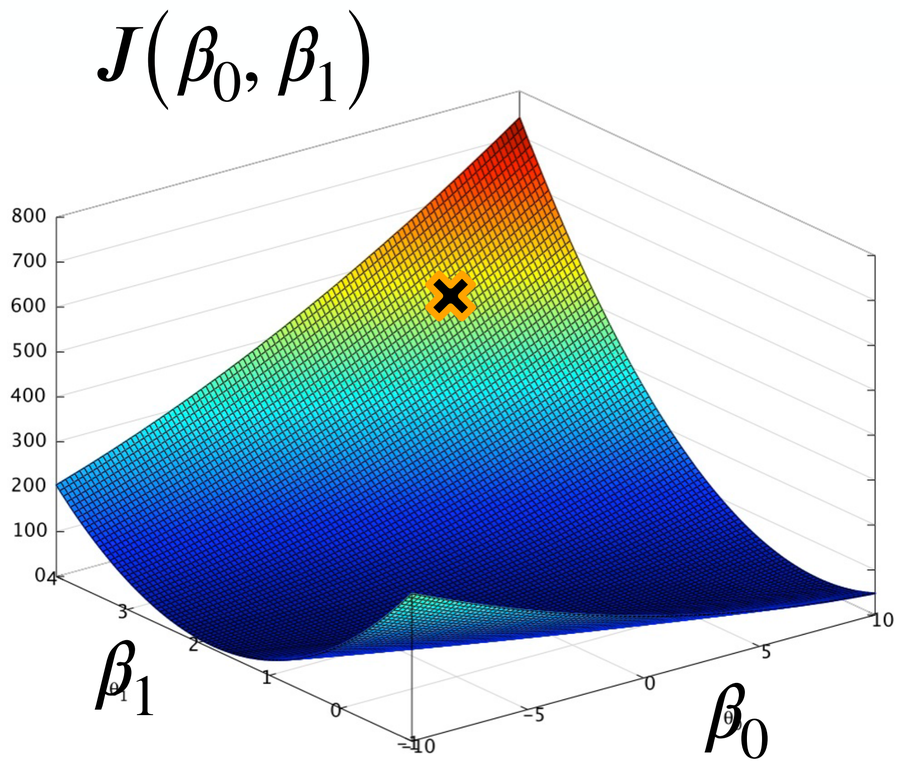
Gradient Descent with Linear Regression

- Now imagine there are two parameters ()
- This is a more complicated surface on which the minimum must be found
- How can we do this without knowing what looks like?



Gradient Descent with Linear Regression

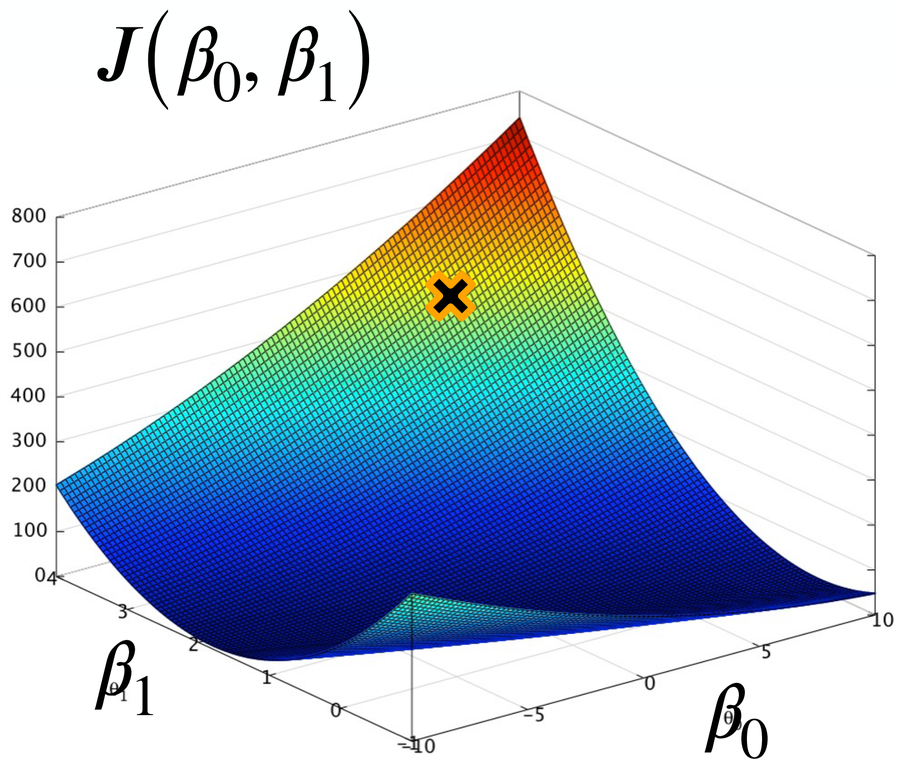
- Compute the gradient, , which points in the direction of the biggest increase!
- - (negative gradient) points to the biggest decrease at that point!



Gradient Descent with Linear Regression

- The gradient is the a vector whose coordinates consist of the partial derivatives of the parameters

$$\nabla J(\beta_0, \ldots, \beta_n) = \; < \frac{\partial J}{\partial \beta_0}, \ldots, \frac{\partial J}{\partial \beta_n} >$$

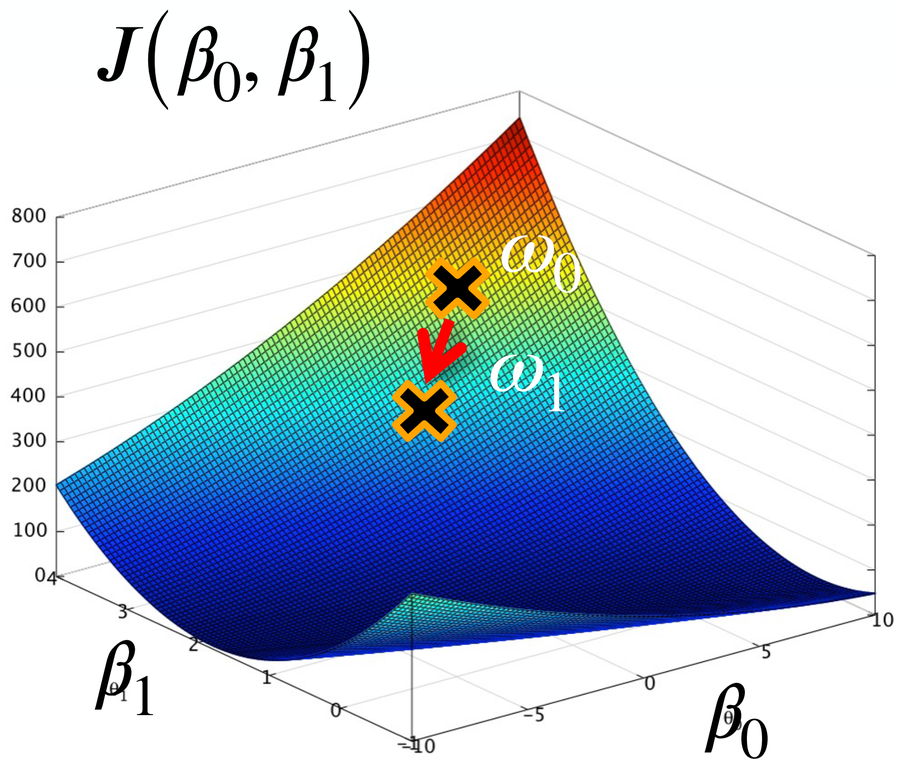


Gradient Descent with Linear Regression

- Then use the gradient and the cost function to calculate the next point () from the current one (

$$\omega_1 = \omega_0 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

- The learning rate () is a tunable parameter that determines step size

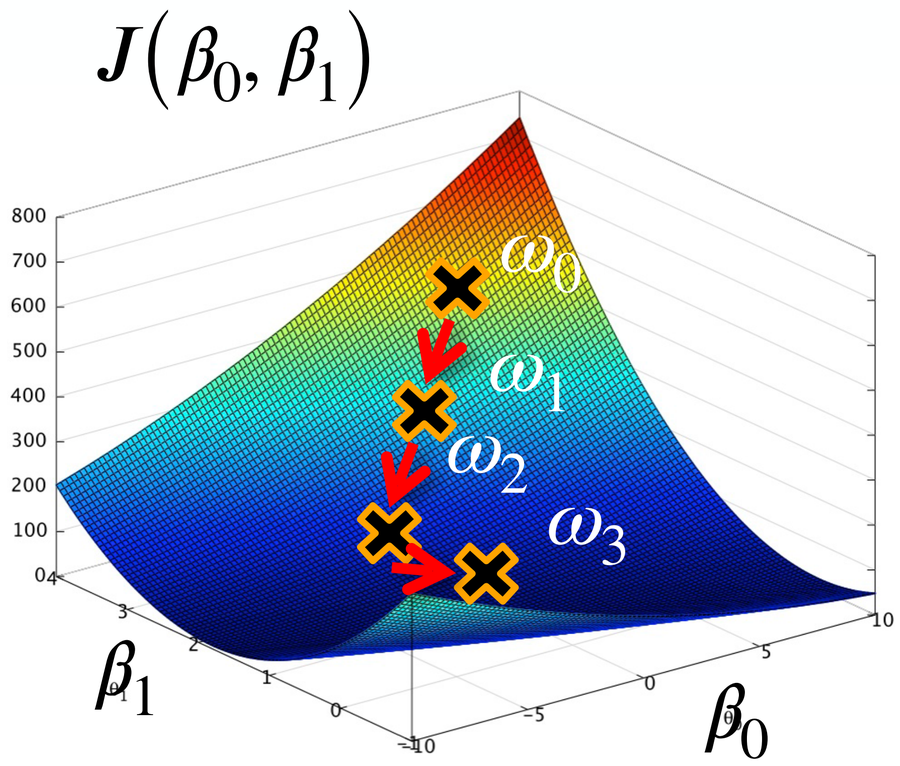


Gradient Descent with Linear Regression

- Each point can be iteratively calculated from the previous one

$$\omega_2 = \omega_1 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

$$\omega_3 = \omega_2 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

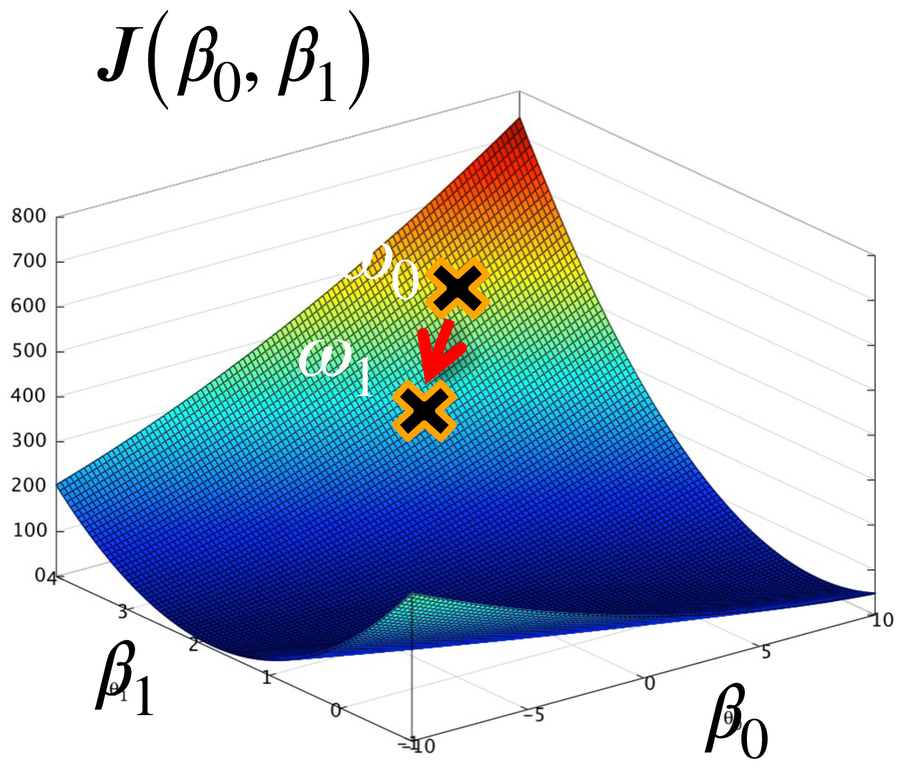


Stochastic Gradient Descent

- Use a single data point to determine the gradient and cost function instead of all the data

$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\cancel{\sum_{i=1}^{m}}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$

$$\boxed{\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\left(\left(\beta_0 + \beta_1 x_{obs}^{(0)}\right) - y_{obs}^{(0)}\right)^2}$$



Stochastic Gradient Descent

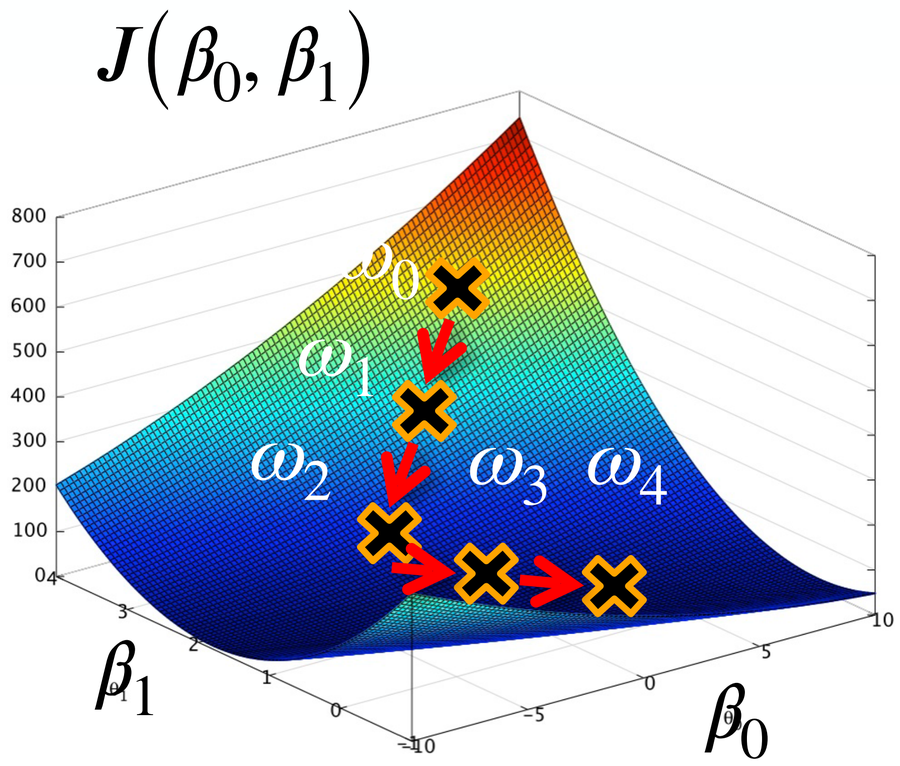
- Use a single data point to determine the gradient and cost function instead of all the data

$$\omega_1 = \omega_0 - \alpha \nabla \frac{1}{2}\left(\left(\beta_0 + \beta_1 x_{obs}^{(0)}\right) - y_{obs}^{(0)}\right)^2$$

$$\ldots$$

$$\omega_4 = \omega_3 - \alpha \nabla \frac{1}{2}\left(\left(\beta_0 + \beta_1 x_{obs}^{(3)}\right) - y_{obs}^{(3)}\right)^2$$

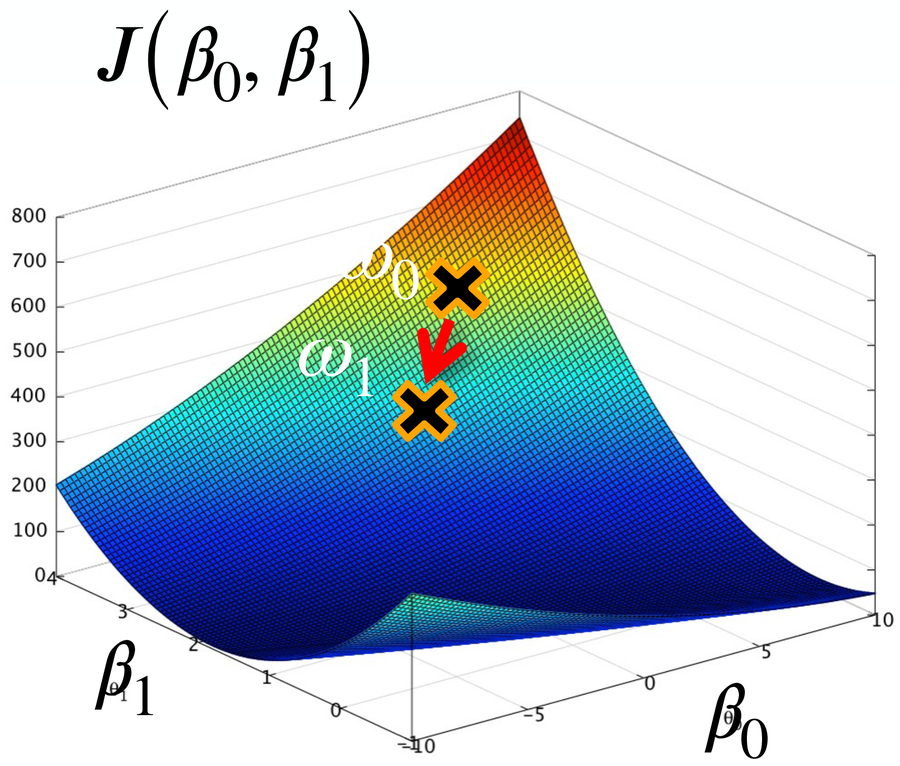
- Path is less direct due to noise in single data point—"stochastic"



Mini Batch Gradient Descent

- Perform an update for every training examples

$$\omega_1 = \omega_0 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{n} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$



Best of both worlds:

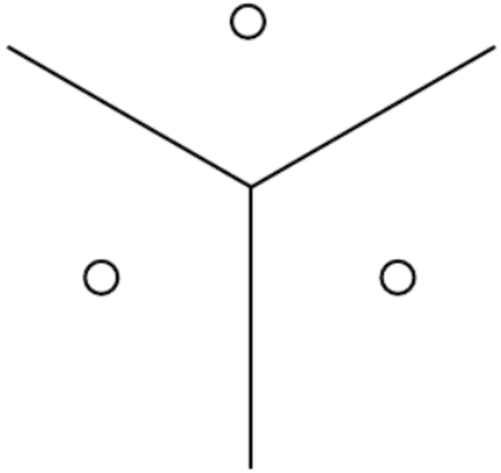
- Reduced memory relative to "vanilla" gradient descent
- Less noisy than stochastic gradient descent

Motivation

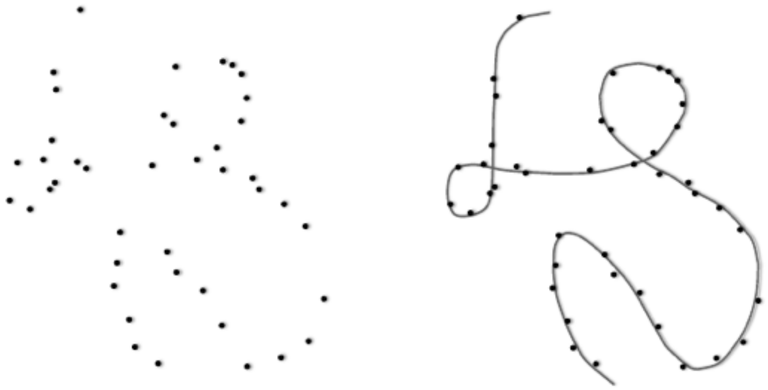
Motivation

- Curse of dimensionality

- Local Constancy & Smoothness Regularization



- Manifold Learning



Implementation

Let’s start with regression

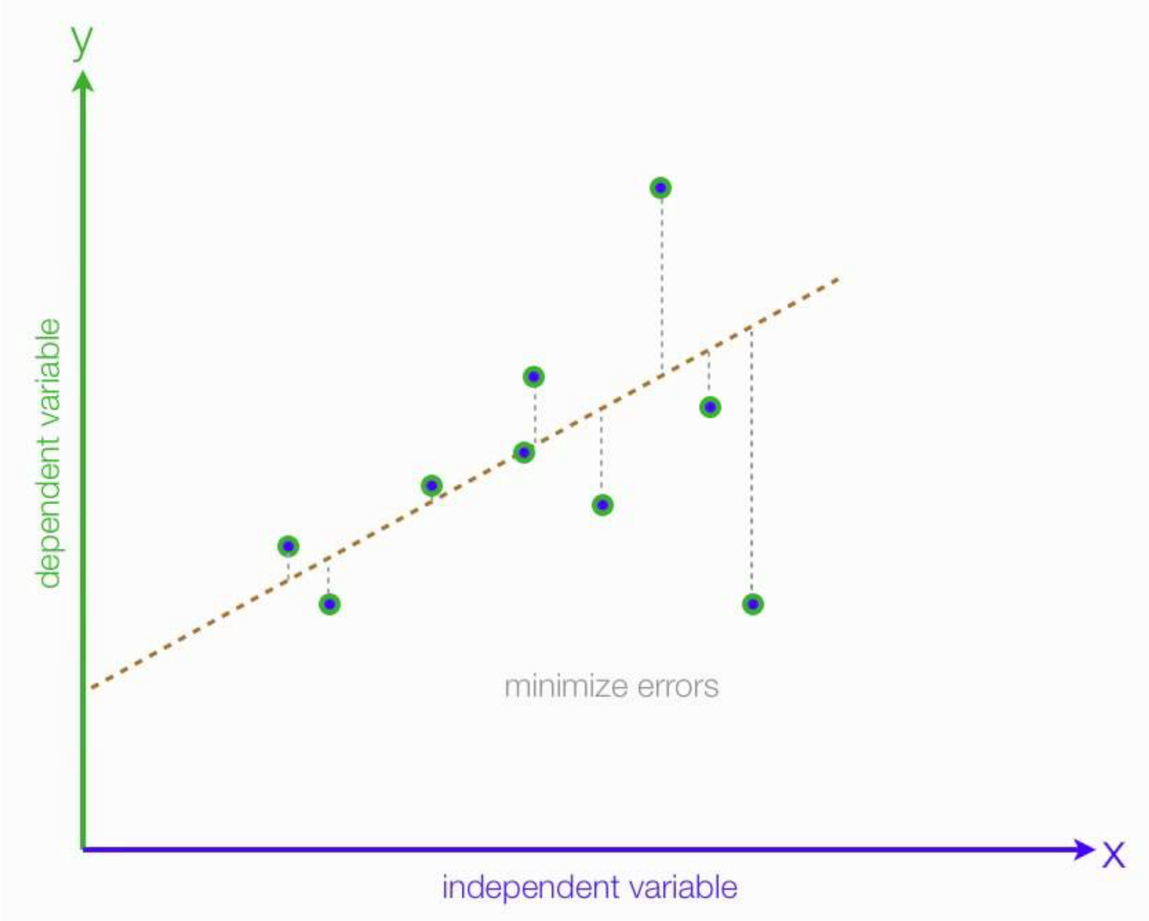
Regression

- Modeling the relationship between a scalar response and one or more explanatory variables.
    - One explanatory variables = **linear regression**
    - Multiple explanatory variables = **multivariate regression**
- Predictions of…
    - Price of an item, based upon the market analysis
    - Salary, based upon company salaries, their experience, ..
    - Score of a sporting event, based on team analytics
    - GDP growth, based on consumer data

Regression

How does it work?

- Linear Regression
  - Least Squares method
    - Published by Legendre (1805) and Gauss (1809)
    - Variations: Ordinary, Weighted, Generalized

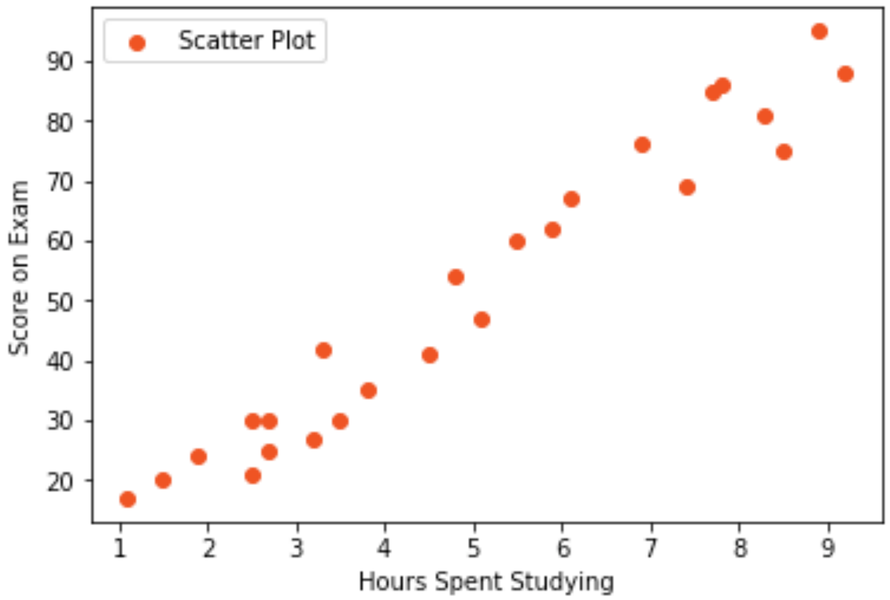


$$\mathrm{MSE}_{\text{test}} = \frac{1}{m}\sum_i (\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})})_i^2$$

Source: *Linear Regression Analysis: Theory and Computing*, Xin Yan

Regression

Implementation - Toy Application



Hours	Scores
2.5	21
5.1	47
3.2	27
8.5	75
3.5	30
1.5	20
9.2	88
5.5	60
8.3	81
2.7	25
7.7	85
5.9	62
4.5	41
3.3	42
1.1	17
8.9	95
2.5	30
1.9	24
6.1	67
7.4	69
2.7	30
4.8	54
3.8	35
6.9	76
7.8	86

Datasets

Useful if review needed

- UCI Machine Learning Repository
- Scikit-learn toy datasets
- Kaggle - Logistic Regression for Titanic dataset
- Princeton datasets

TensorFlow

What is TensorFlow?

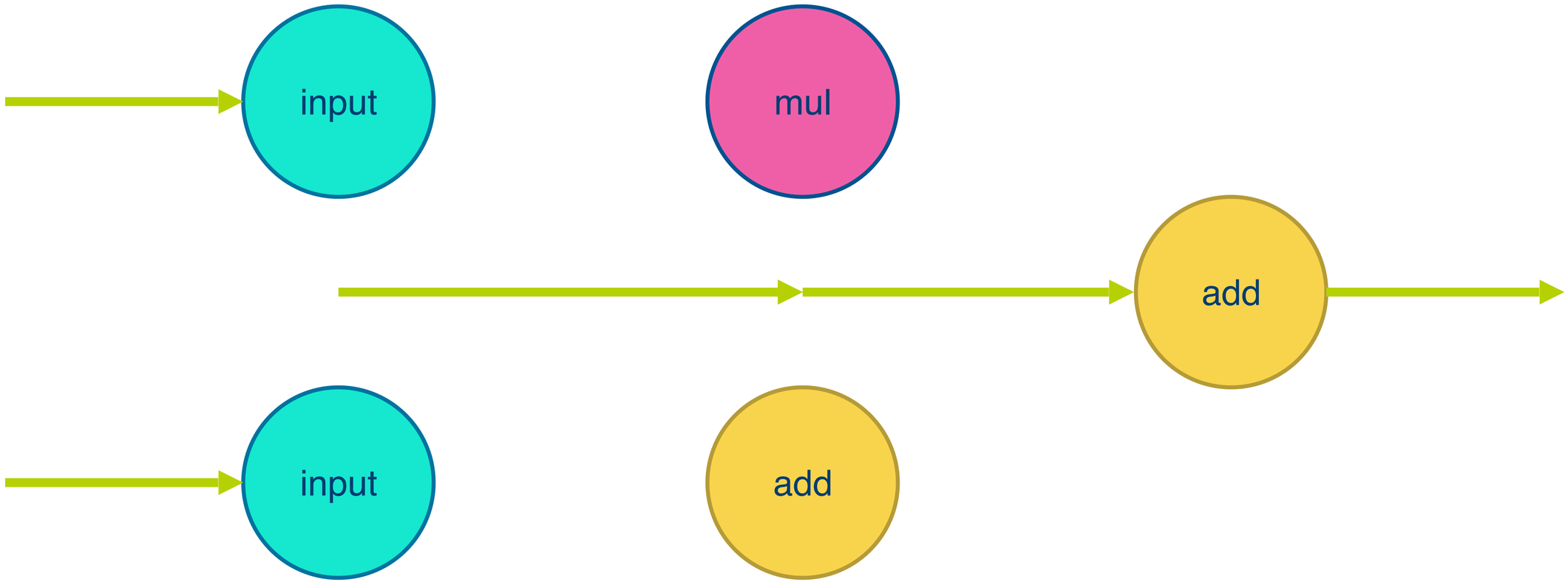
- A **framework** provided by Google for math using **computation graphs**.
    - Specific features for machine learning
- **Primary interface**: Python
    - Integrates with NumPy
- **Implementation**: C++ & CUDA
    - Designed to be flexible, scalable, & deployable

TF History

- Google's successor to DistBelief
- Timeline:
  - Nov 2015: Open source implementation released
  - Feb 2016: TensorFlow Serving
  - Apr 2016: Distributed runtime
  - Jan 2017: XLA (Accelerated Linear Algebra)
  - Feb 2017: TensorFlow Fold
- *Backwards compatibility promised between releases (i.e. 1.0.0 until 2.0.0)*

https://www.tensorflow.org/api_docs/python/tf

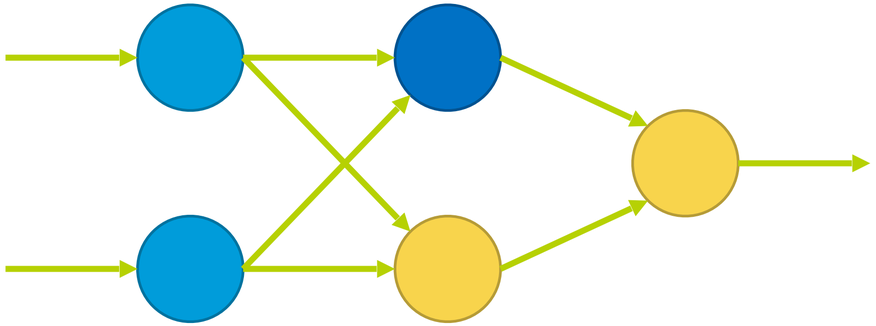
Computation Graphs



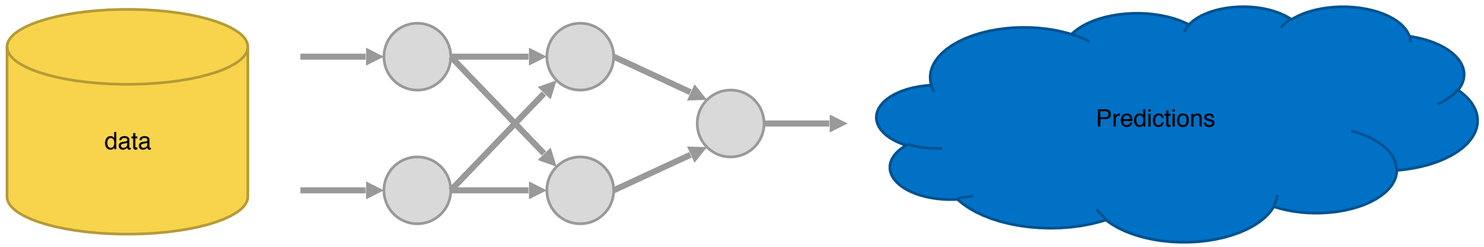
Getting Started: TensorFlow

<u>Two-step programming pattern:</u>

1. Define the computation graph



2. Run the graph



TF 1. Define Computation Graph

TensorFlow API classes:

Graph Container for the operations & tensors..

```
import tensorflow as tf
```

Operation Nodes in the graph, representing computations

```
a = tf.constant( 3.0, name="input1" )
b = tf.constant( 2.0, name="input2" )
```

Or, instead of hard-coding, use a placeholder:

```
a = tf.placeholder( tf.float32, name="input1" )
```

Tensor Edges in the graph, representing data

```
c = tf.add( a, b, name="my_add_op" )
d = tf.multiply( a, c, name="my_mul_op" )
```

TF 2. Run Computation Graph

TensorFlow API classes:

Session objects are dedicated to a single graph for execution.

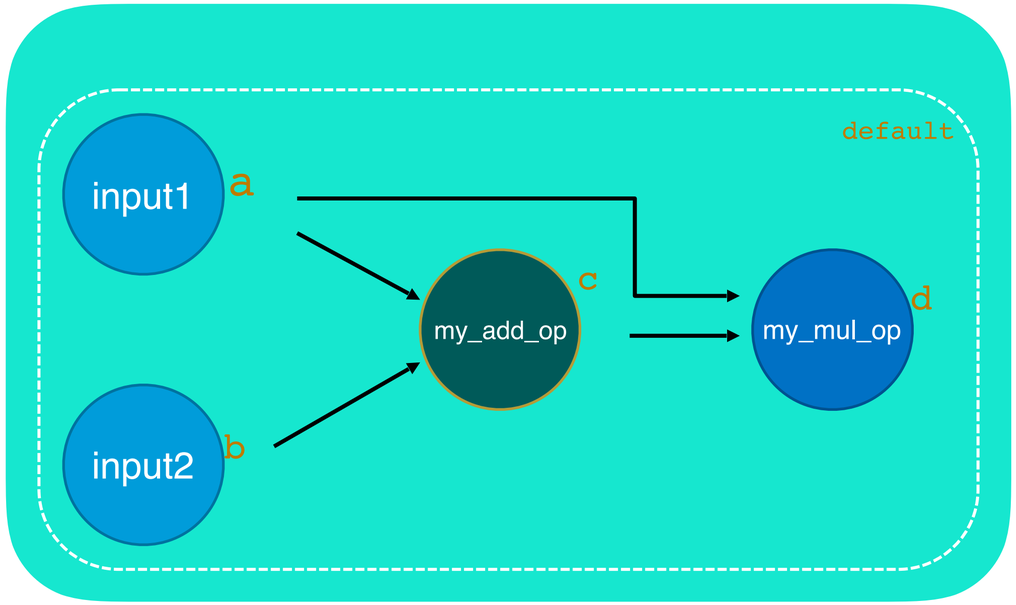
```
sess = tf.Session()
```

Placeholders contain data - a dictionary mapping Tensor keys to numeric values.

```
feed_dict = {a:3.0, b:2.0}
```

Fetches tells TF what outputs we want (a Tensor, Operation, or list of them)

```
out = sess.run( d, feed_dict=feed_dict )
```



TensorFlow Variables

For **stateful information**, use **`Variable`** objects

```
store = tf.Variable( 0.0, name="var" )
```

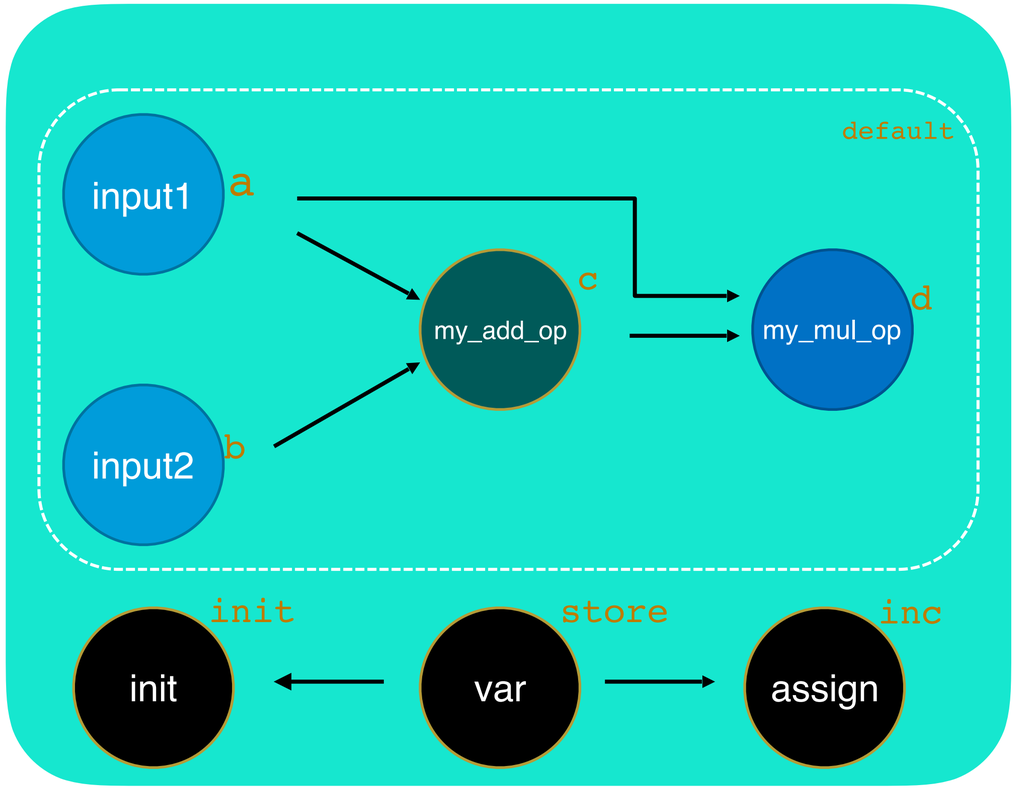
Create operations to update these values.

```
inc = store.assign( store + 1 )
```

Variables are not established in the Session until initialized.

```
init = tf.global_variables_initializer()

sess.run( init )
```



Machine Learning & TensorFlow

Cost Function for Linear Regression

For our previously defined model

$$\hat{y}_i = Wx_i + b$$

Establish **cost function J** via mean squared errors (MSE)

$$\mathrm{J} = \frac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}_i - y_i \right)^2$$

n = # training examples
W = weights
x = input
y = output
b = bias
J = cost function

Minimizing Loss

To minimize this cost, take the derivative:

$$\mathrm{J} = \frac{1}{2n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2$$

$$= \frac{1}{2n}\sum_{i=1}^{n}\left((Wx_i + b) - y_i\right)^2$$

$$\frac{\partial J}{\partial W} = \frac{1}{n}\sum_{i=1}^{n} x_i\left(\hat{y}_i - y_i\right)$$

$$\frac{\partial J}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)$$

n = # training examples
W = weights
x = input
y = output
b = bias
J = cost function

Minimizing Loss

In linear regression, we can use linear algebra..

$$W = (X^T X)^{-1} X^T Y$$

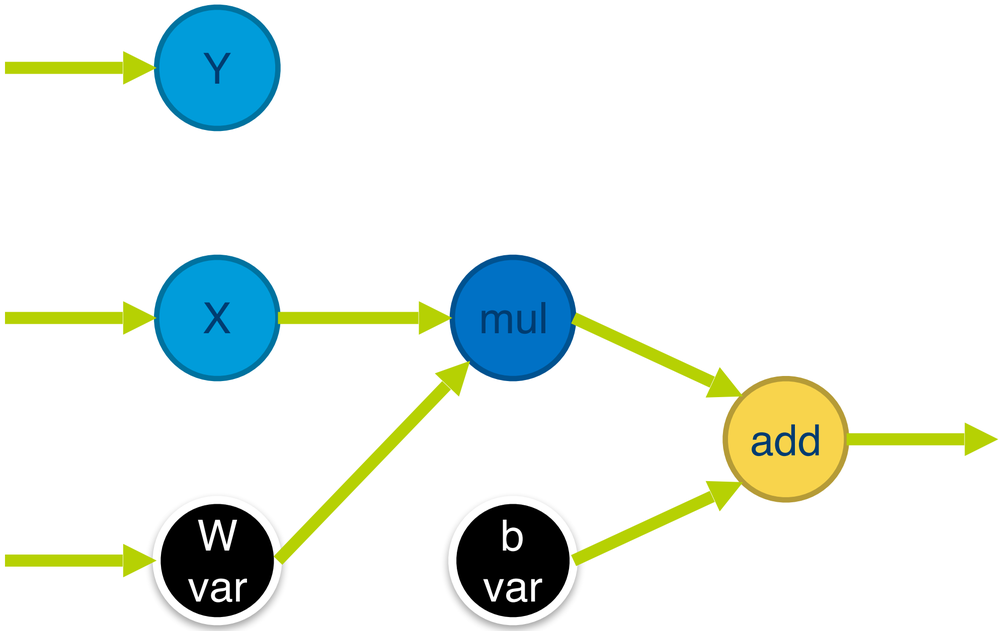
Problems:

- For large input (X), computing the inverse is difficult
- Inverse function is ill-conditioned
- Not all models will have a closed-form solution

n = # training examples
W = weights
x = input
y = output
b = bias
J = cost function

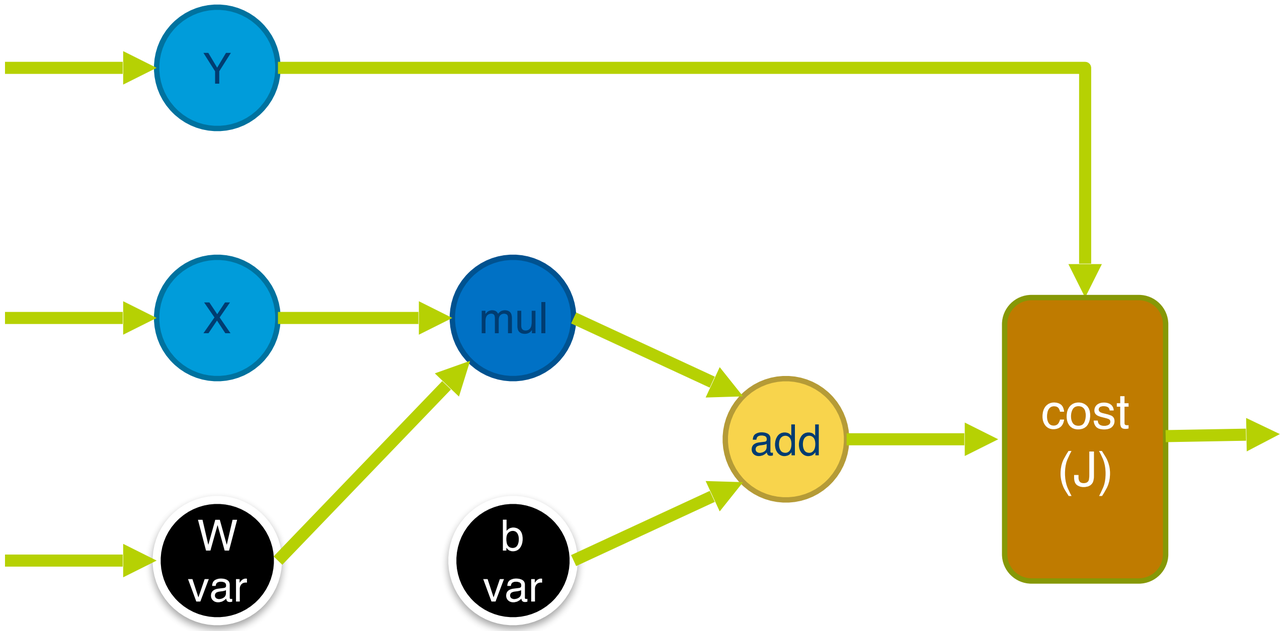
Process of gradient decent (computation graph)

1. Start with our basic model



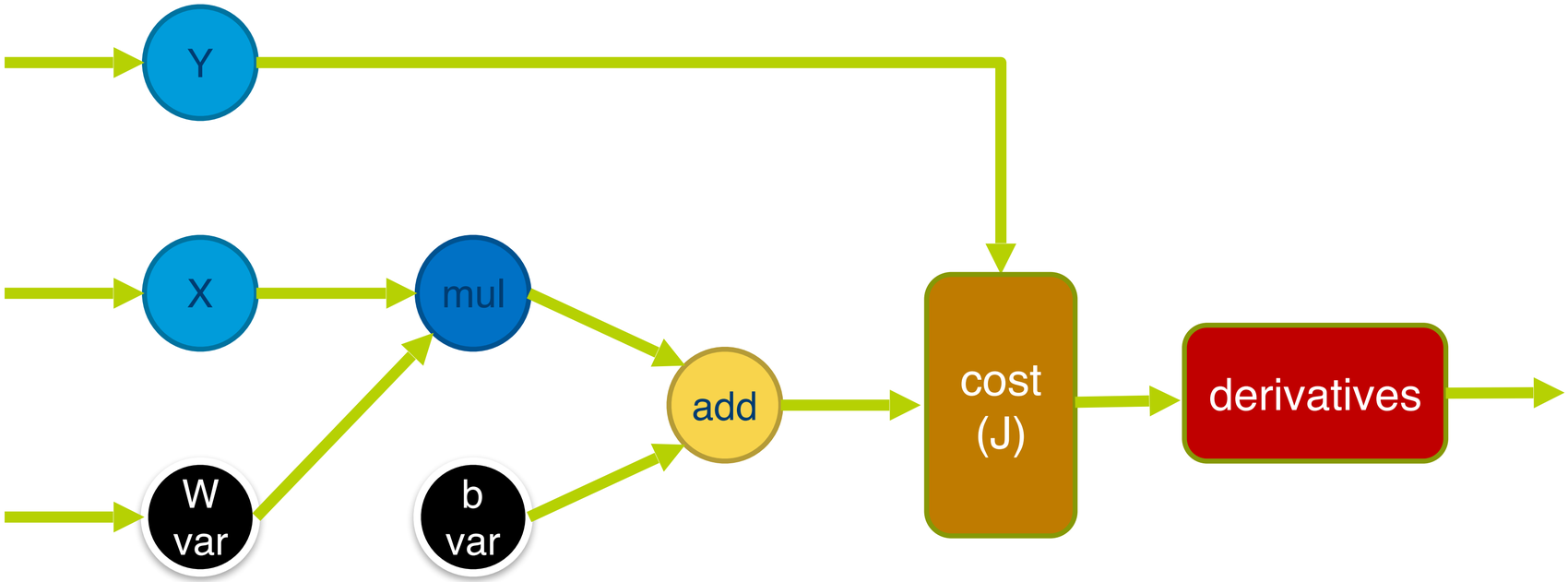
Process of gradient decent (computation graph)

2. Define our cost function
   - In this case `tf.square(add – y)`
   - Built in: `tf.squared_difference(add, y)`



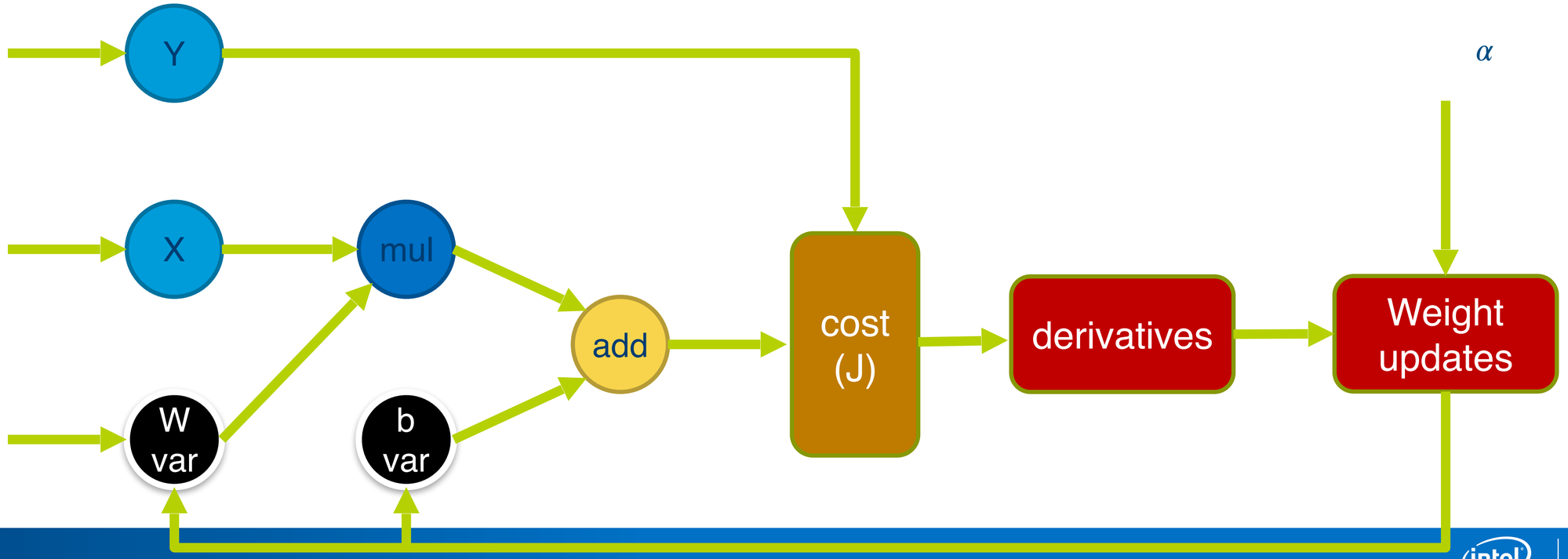
Process of gradient decent (computation graph)

3. Get the derivatives of the cost w.r.t Variables
   - Sum over all examples =



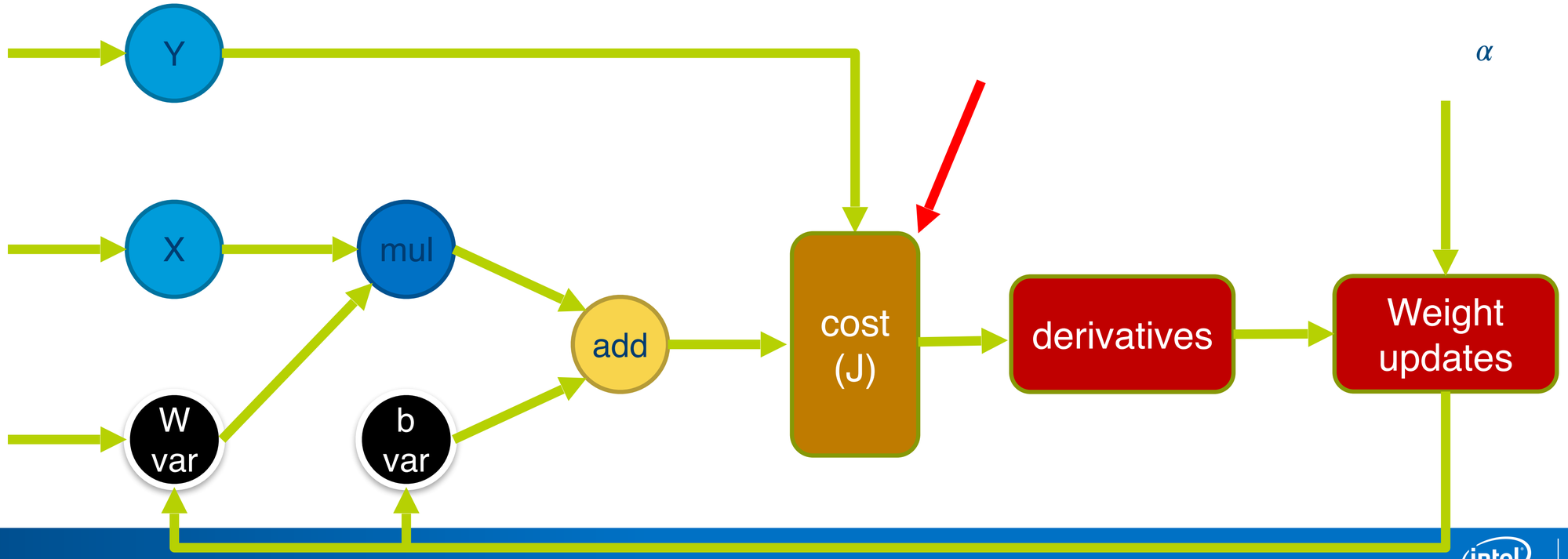
Process of doing gradient decent (computation graph)

4. Use derivatives and learning rate to update Variables



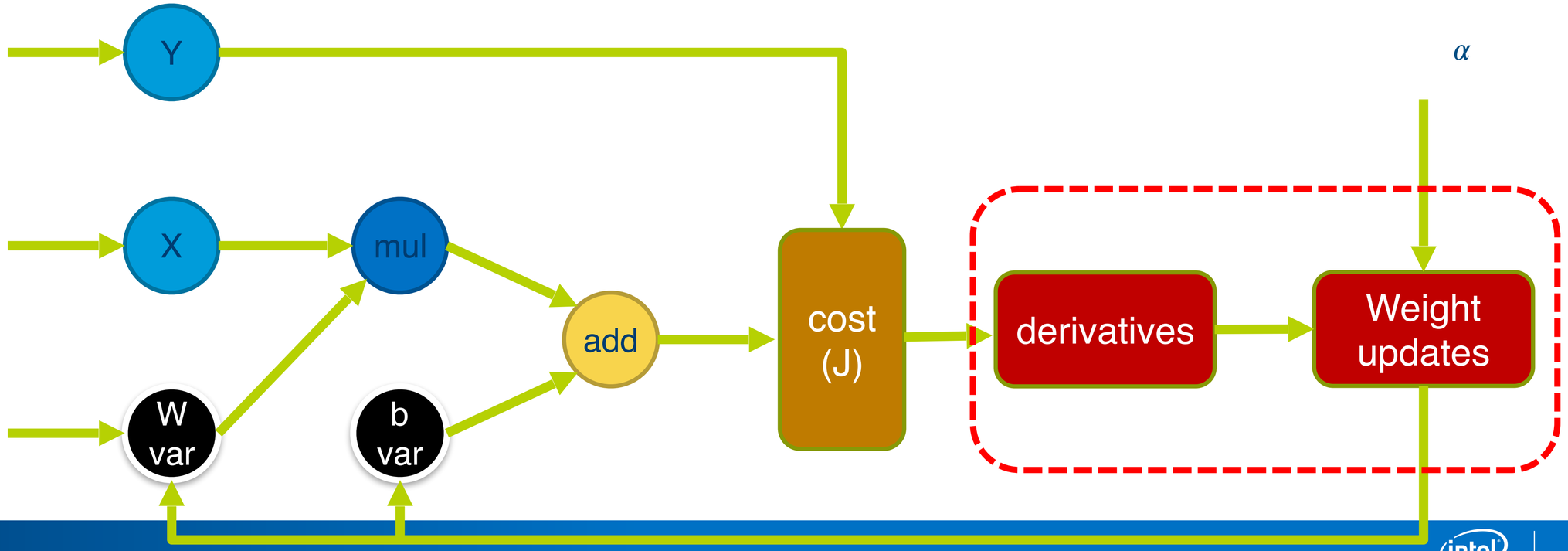
Cost needs to be defined like other operations in TF

```
cost = tf.reduce_mean(tf.squared_difference(add, Y))
```



Derivatives and updates use a `tf.Optimizer`

`Optimizer` computes derivatives and applies them to **`Variables`**



Summary

Lesson 12:
Introduction to Deep Learning

- Motivation for Deep Learning
- Neural Networks
- Evaluation of learning models
- Implementation (intro)

Next week:

- Optimization
- Training
- Regularization