Intel Nervana™ AI Academy

MACHINE LEARNING FUNDAMENTALS

THINGS TO COVER IN SLIDES

Gradient descent

Linear regression

Logistic regression

Training/validation/test splits

Loss function, derivatives

Full batch, mini batch/stochastic gradient descent

Regularization

Epoch

Supervised vs unsupervised training

TensorFlow: Optimizer class, global step

KEY THINGS TO TAKE AWAY FROM TODAY

Loss functions

Gradient descent

Automatic differentiation

WHAT IS MACHINE LEARNING?

MACHINE LEARNING

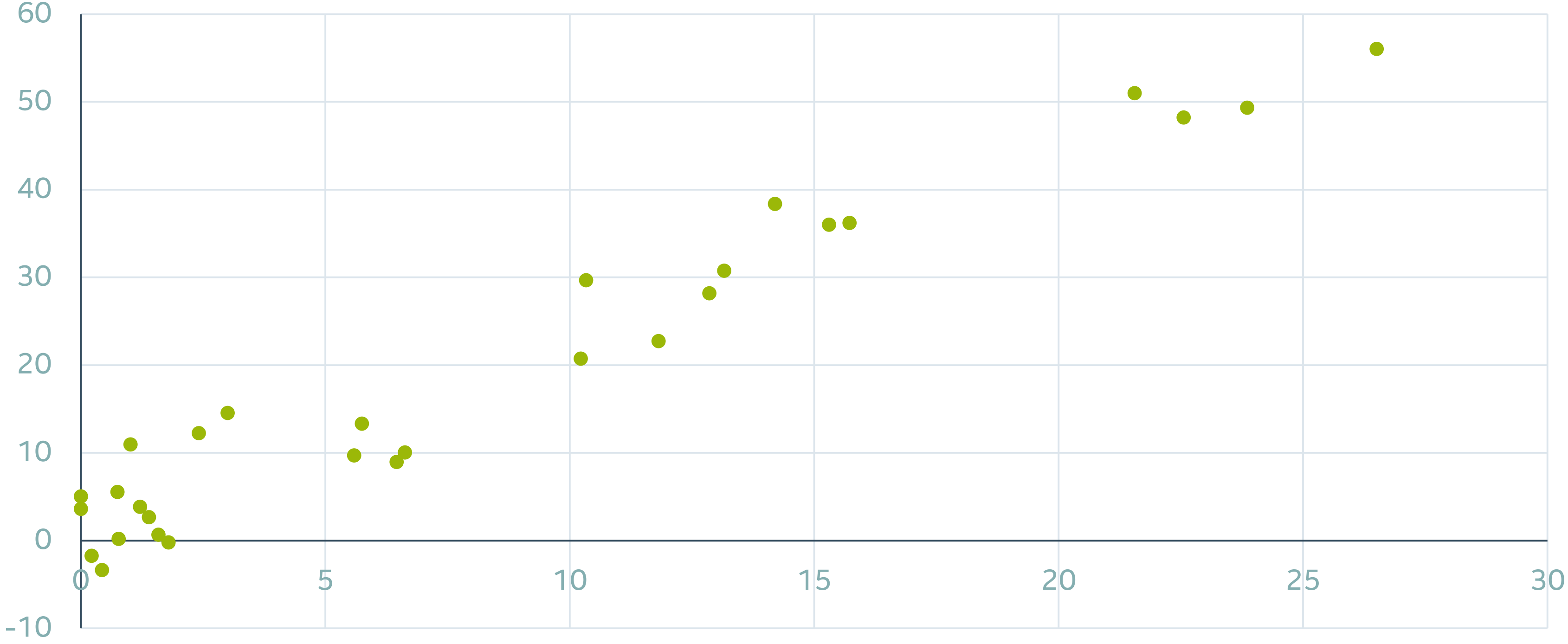
Abstractly:

Giving computers the ability to learn automatically

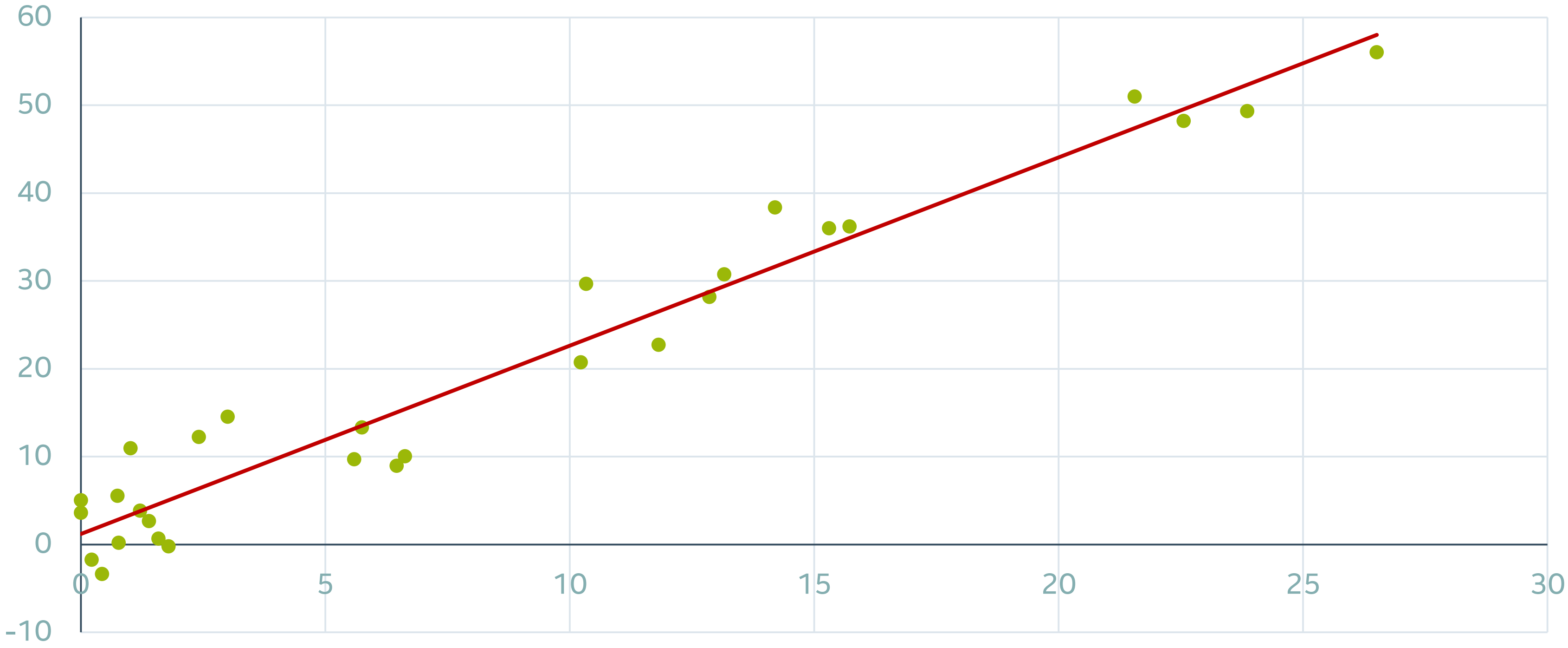
Concretely:

Using math/stats to estimate a model by using data

REMEMBER EXCEL?



REMEMBER EXCEL?



KINDS OF MACHINE LEARNING TASKS

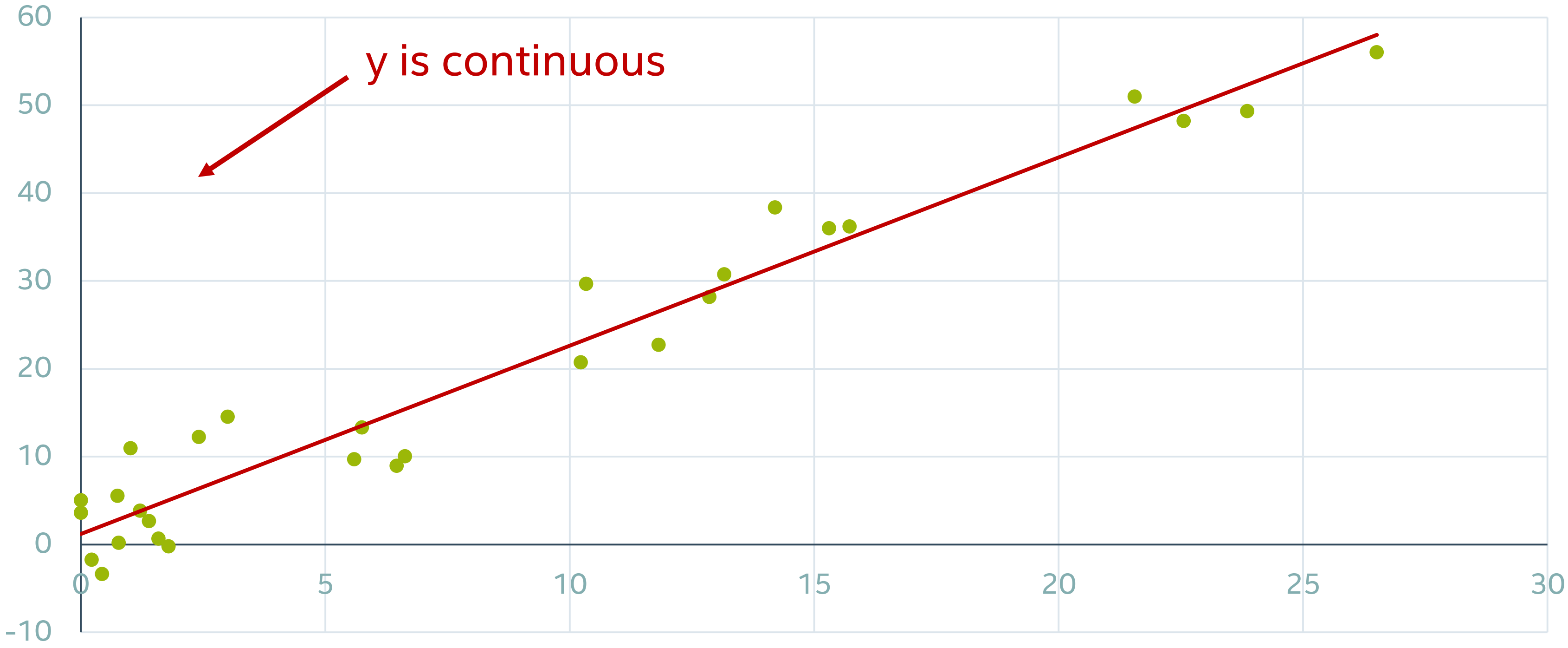
Regression: predict continuous valued output

- House $ based on attributes
- How far to the left or right to turn a car

Classification: predict discreet categories of output

- Which kind of animal is in a picture?
- What sort of anomaly is in an x-ray scan, if any?

A REGRESSION TASK



KINDS OF MACHINE LEARNING METHODS

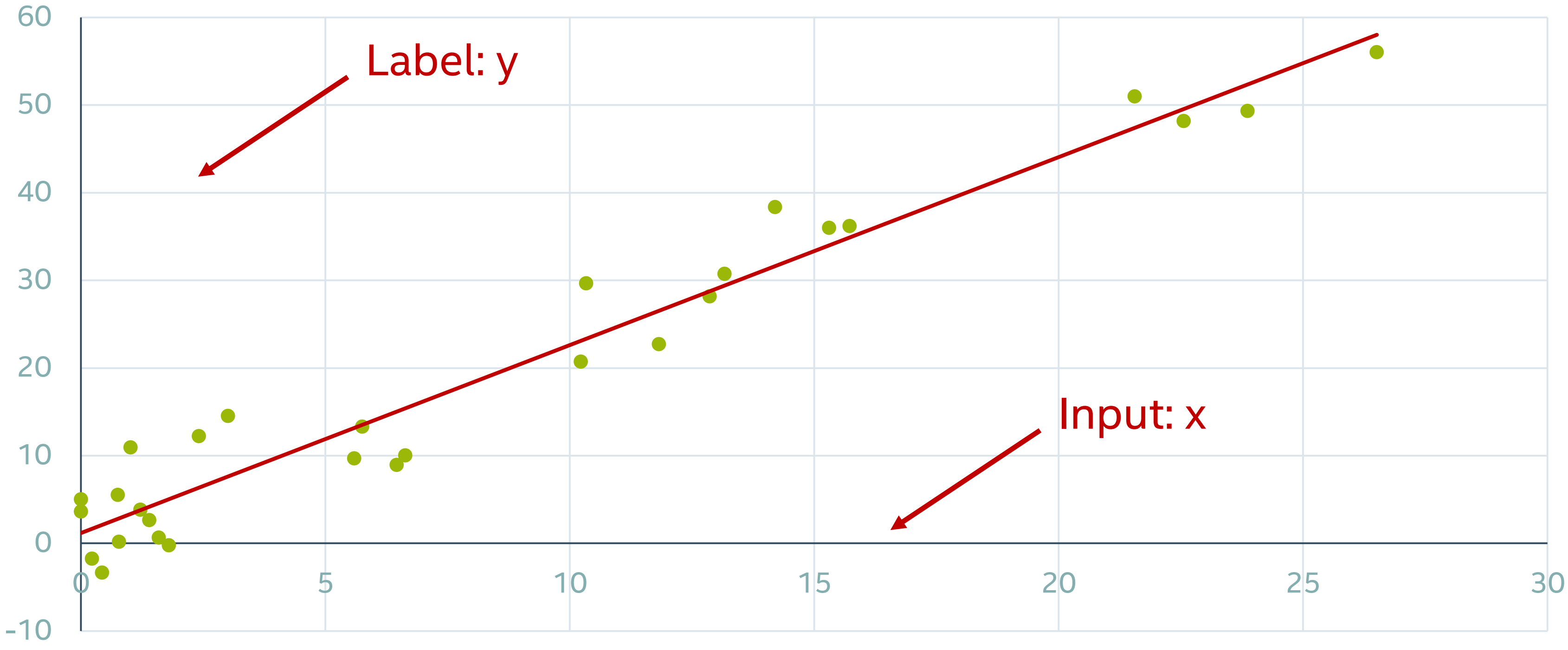
Supervised: train model using specific labeled or known data points

- The model is trying to hit a target
- Targets or Labels:
  - Price of a house
  - The correct steering angle for a car
  - Category of an item
  - Boolean: Yes/No, Risk/Safe

Unsupervised: train model without labels

- Model is trying to find patterns of input data
- Examples:
  - Clustering
  - Autoencoders

A IS SUPERVISED LEARNING TASK

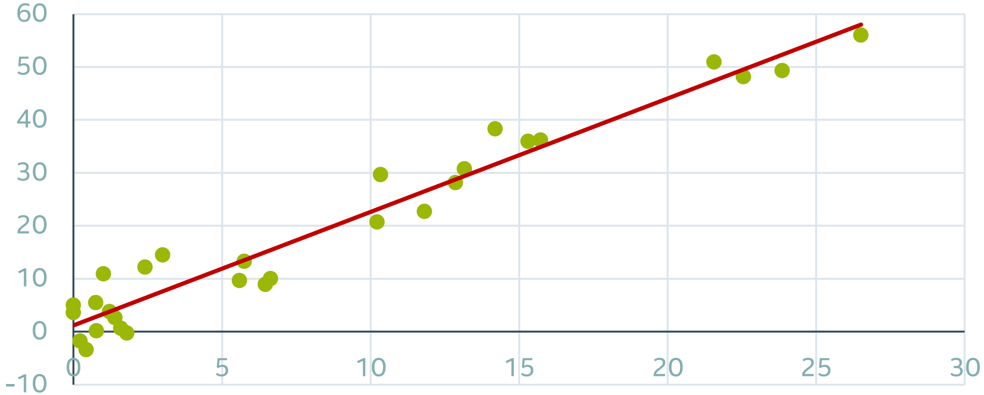


LINEAR REGRESSION

Best fit line: $\hat{y} = Wx + b$

- Line 'through the middle' of a scatter plot
- How do we define "best fit"?

Supervised regression task



COST FUNCTION: J

Idea:

Create a measure model wrongness: a cost function or J and use math to minimize how wrong we are

Question: how to determine what cost function to use?

Answer A:

- Use statistical theory to find an MLE
- Create measure that empirically works
- Think hard for a long time

Answer B: Reuse the work of smart people

COST FUNCTION FOR LINEAR REGRESSION

Classical: sum of squared errors, or SSE

Sum over all n training examples

$$\hat{y}_i = Wx_i + b; \qquad J = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

A little nicer: mean squared errors: MSE

- Simply divide error by the number of examples
- If # training examples increases, your error doesn't

2 is here to make math nicer soon

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

HOW TO MINIMIZE LOSS?

Use calculus!

Take derivative, find values for W,b that make it equal zero!

$$J = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{2n}\sum_{i=1}^{n}((Wx_i + b) - y_i)^2$$

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

Make both equal 0

HOW TO MAKE DERIVATIVE EQUAL ZERO?

For linear regression: you can use linear algebra to solve exactly

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

W and b are lumped together here

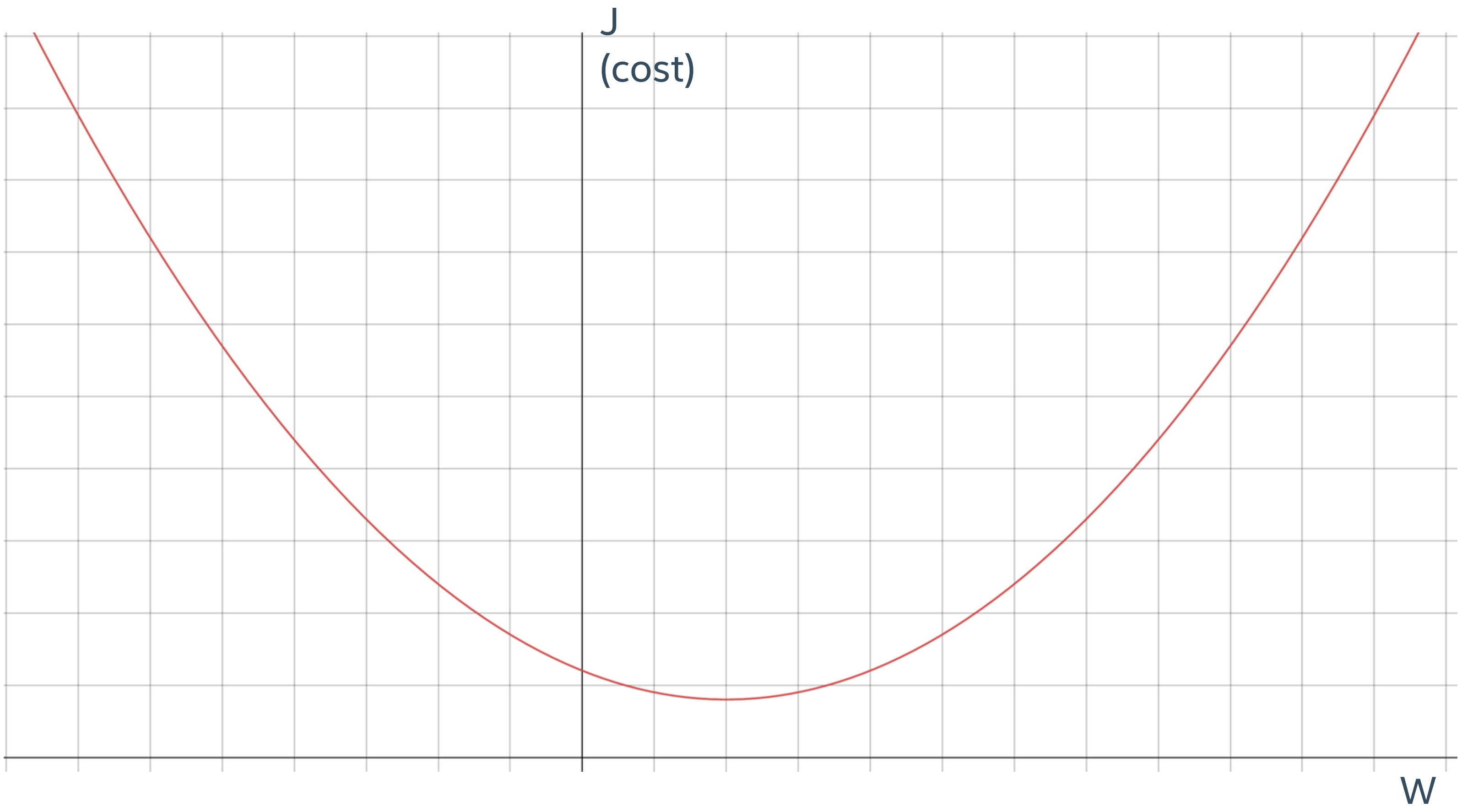
Problems

1. For big X, hard to compute inverse
2. Inverse is ill-conditioned
3. Not all models have a nice closed form linear algebra solution!

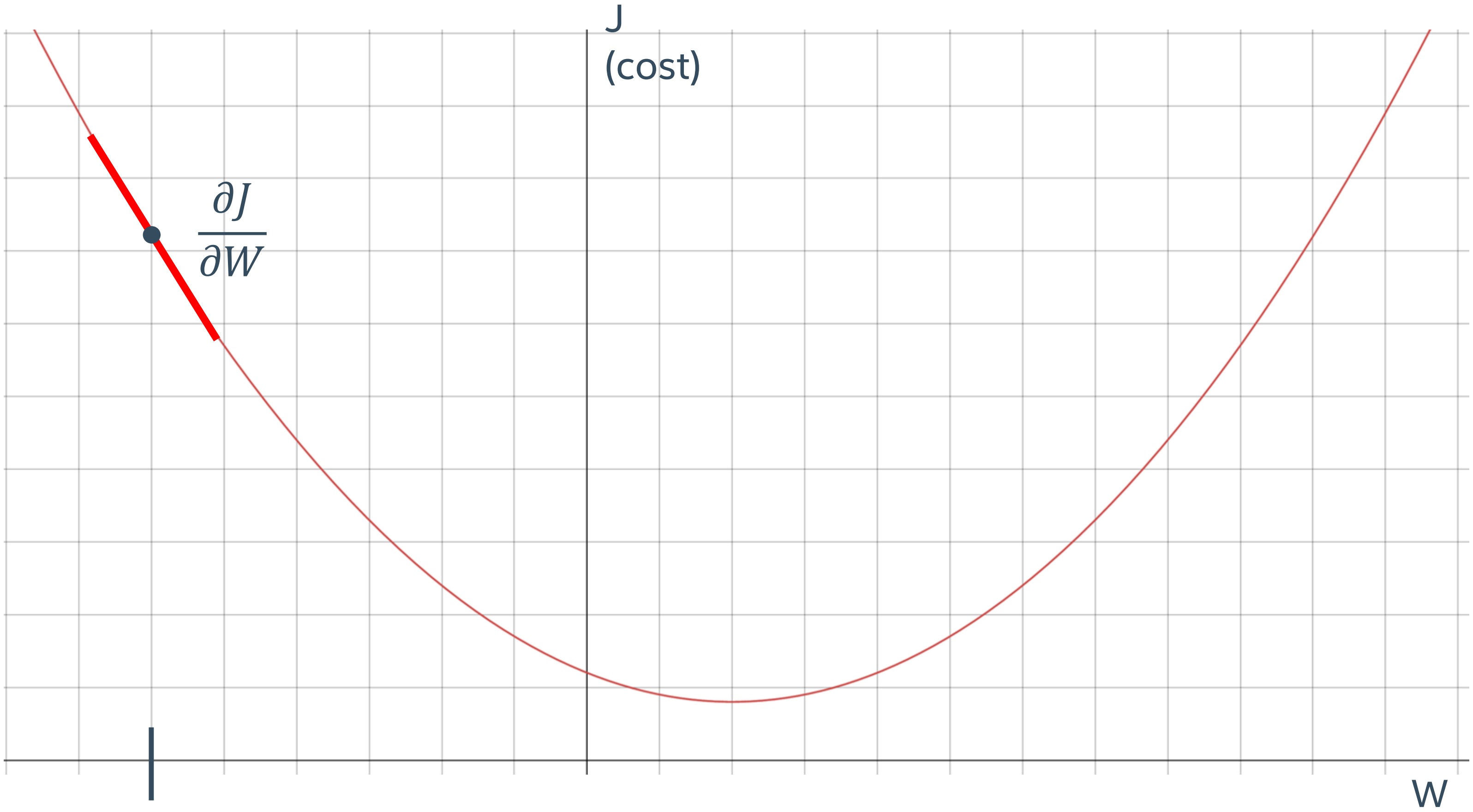
What to do?

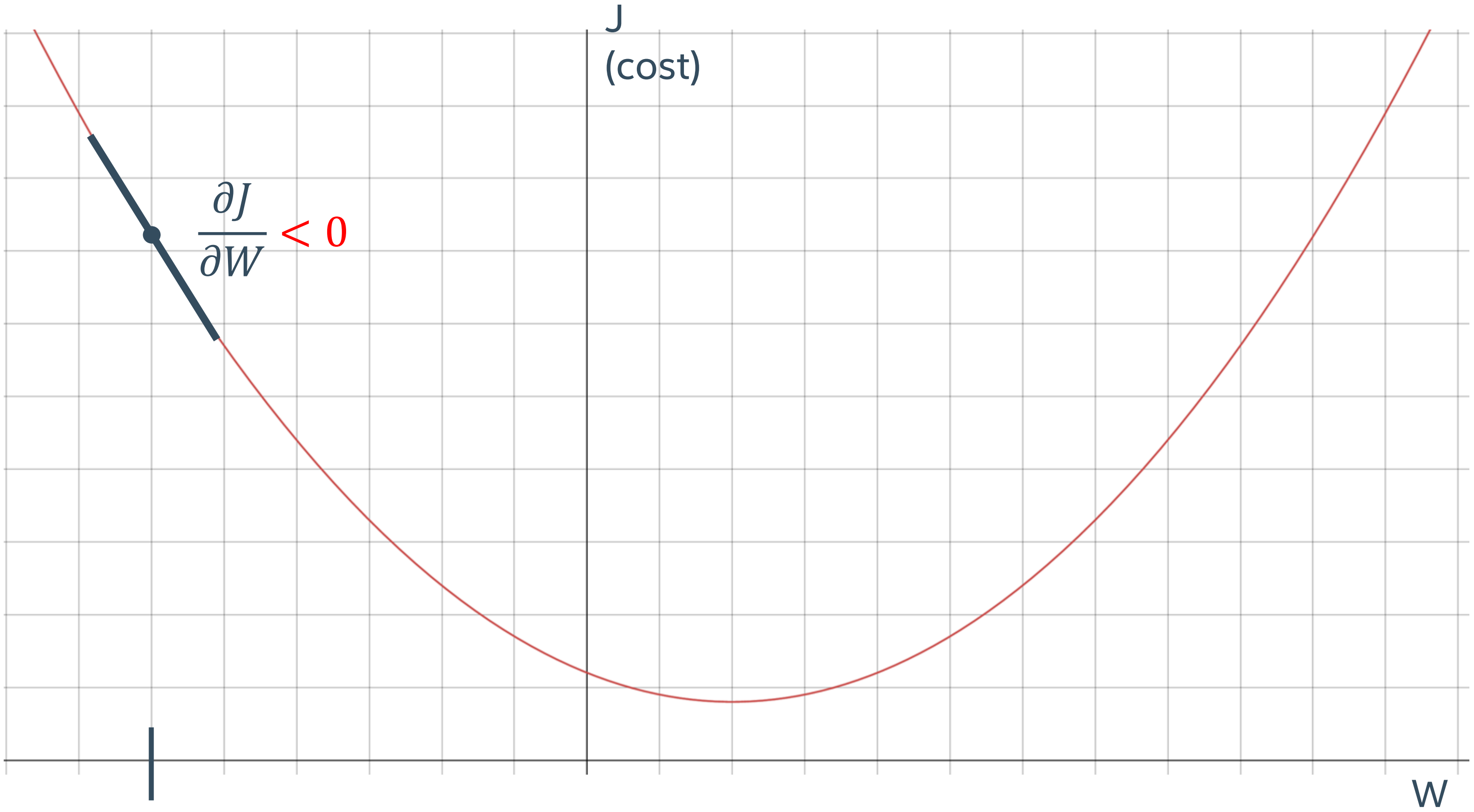
GRADIENT DESCENT

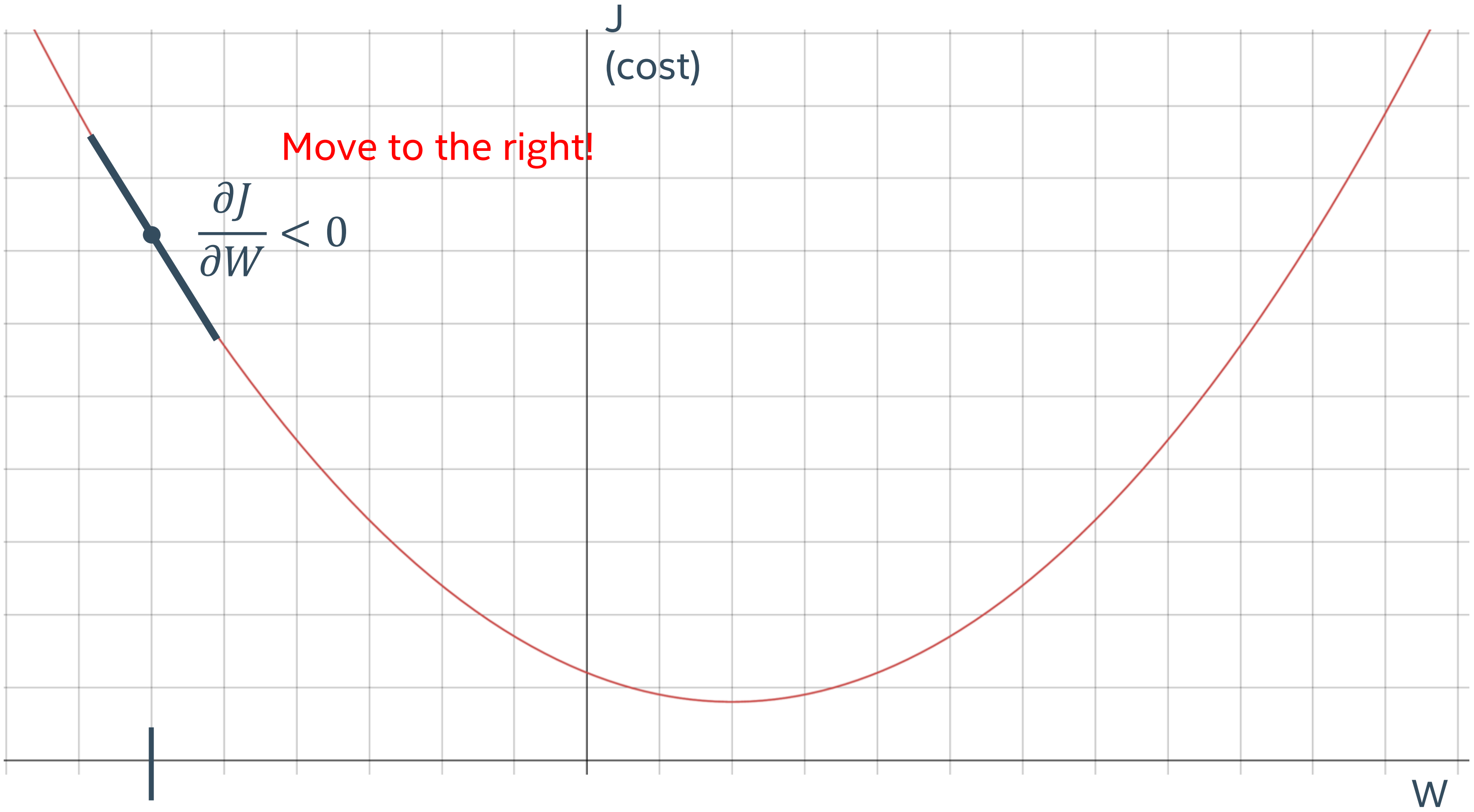
Guess and check for data scientists

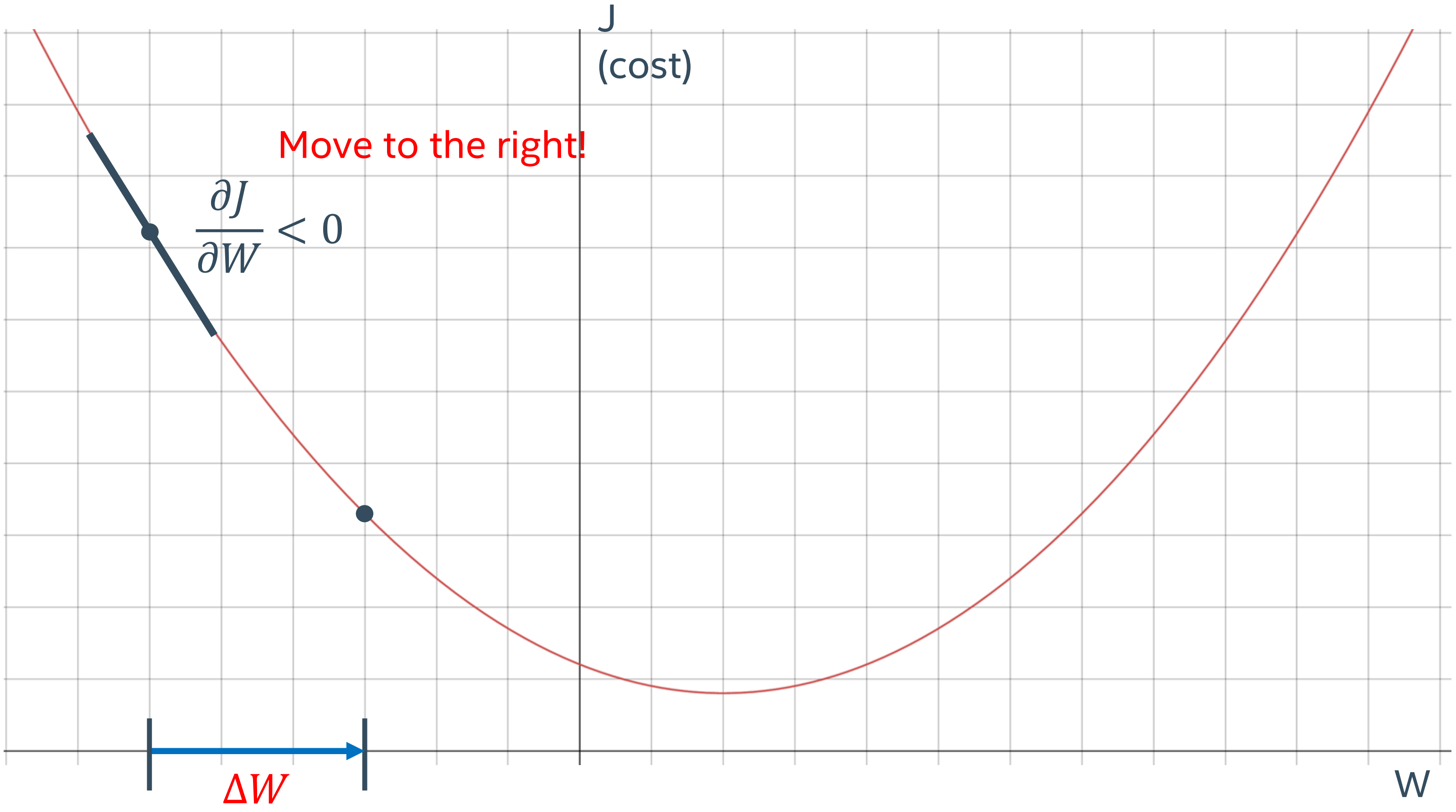

J
(cost)
W

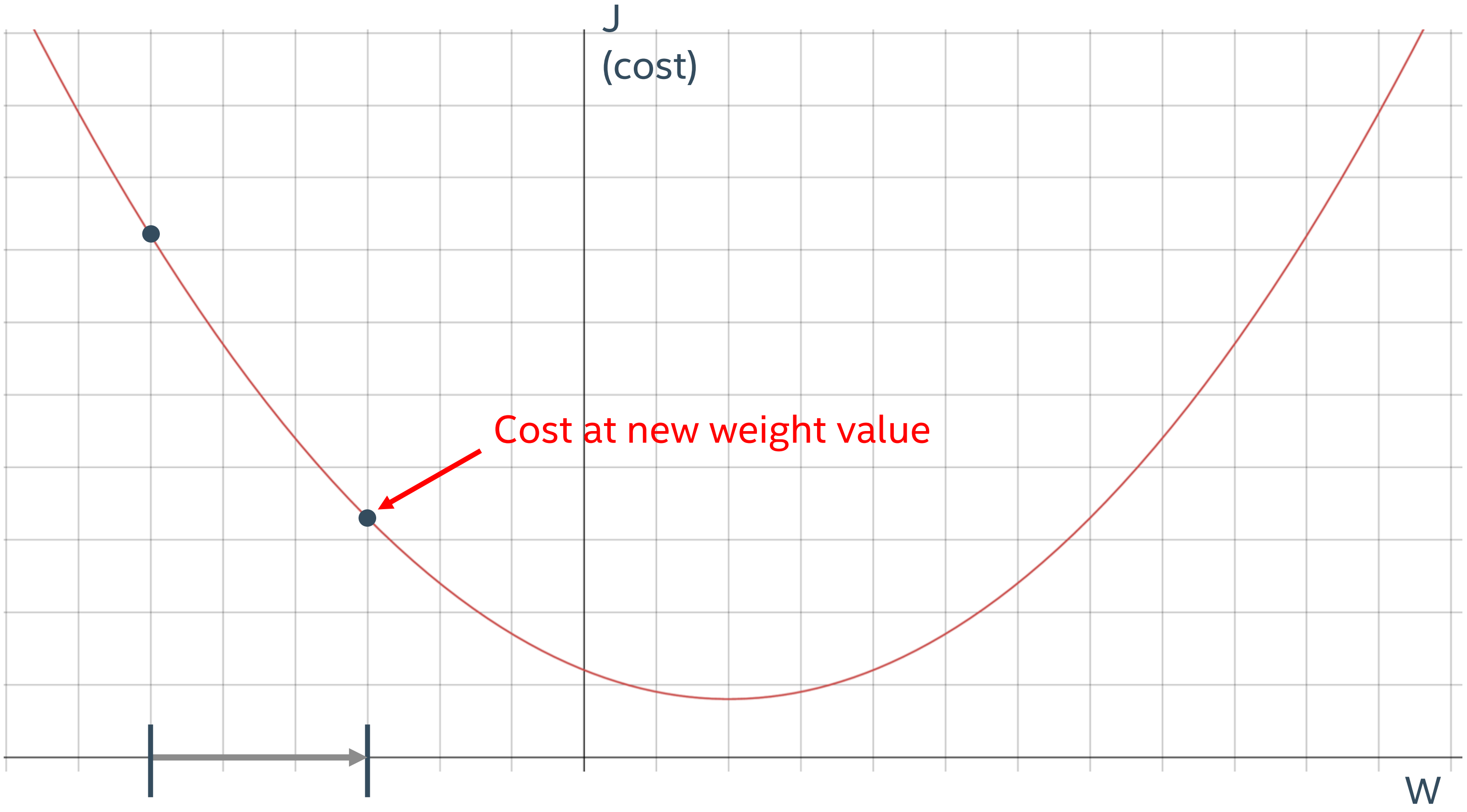

J
(cost)
Random guess put us here
W

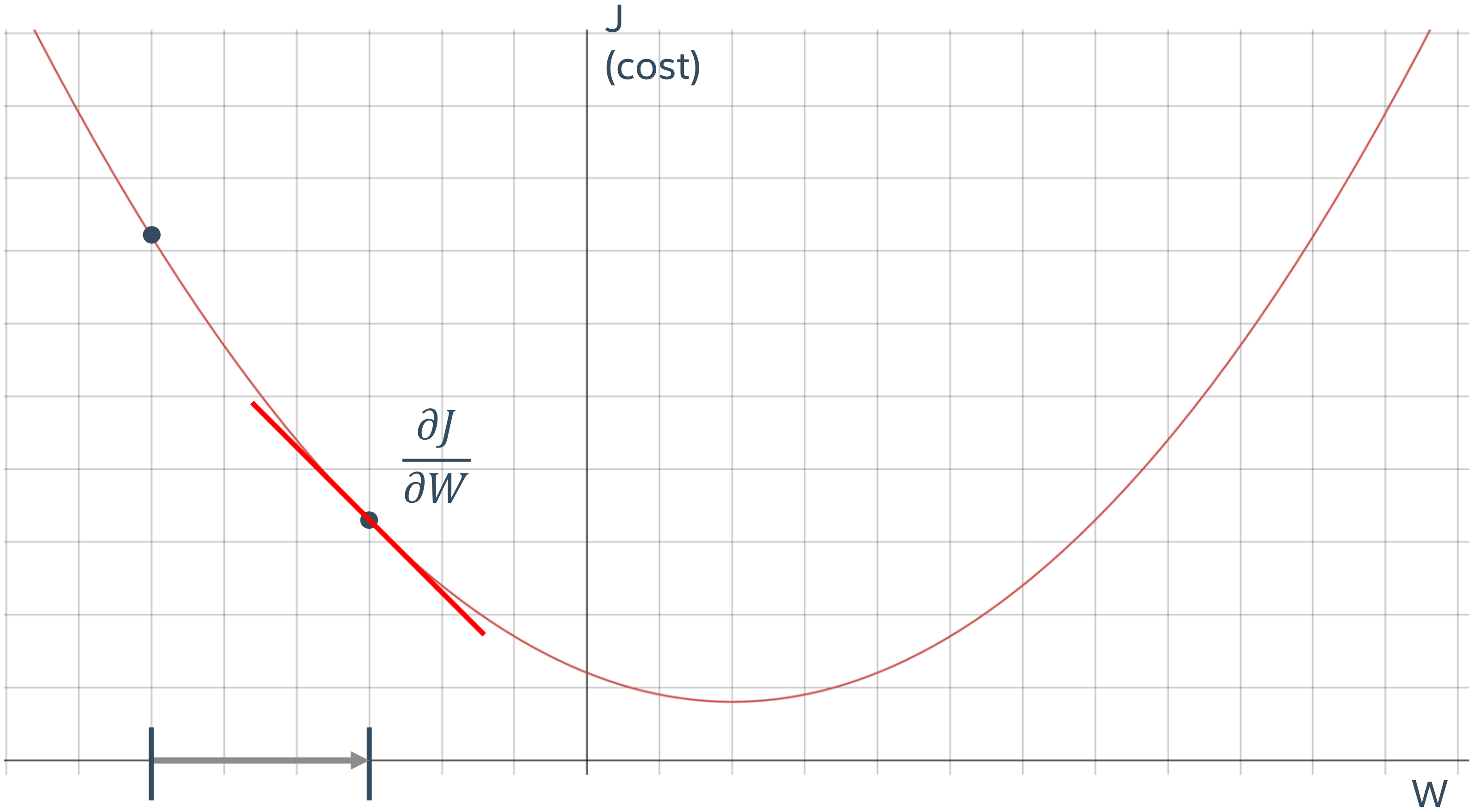

J
(cost)
$\frac{\partial J}{\partial W}$
W

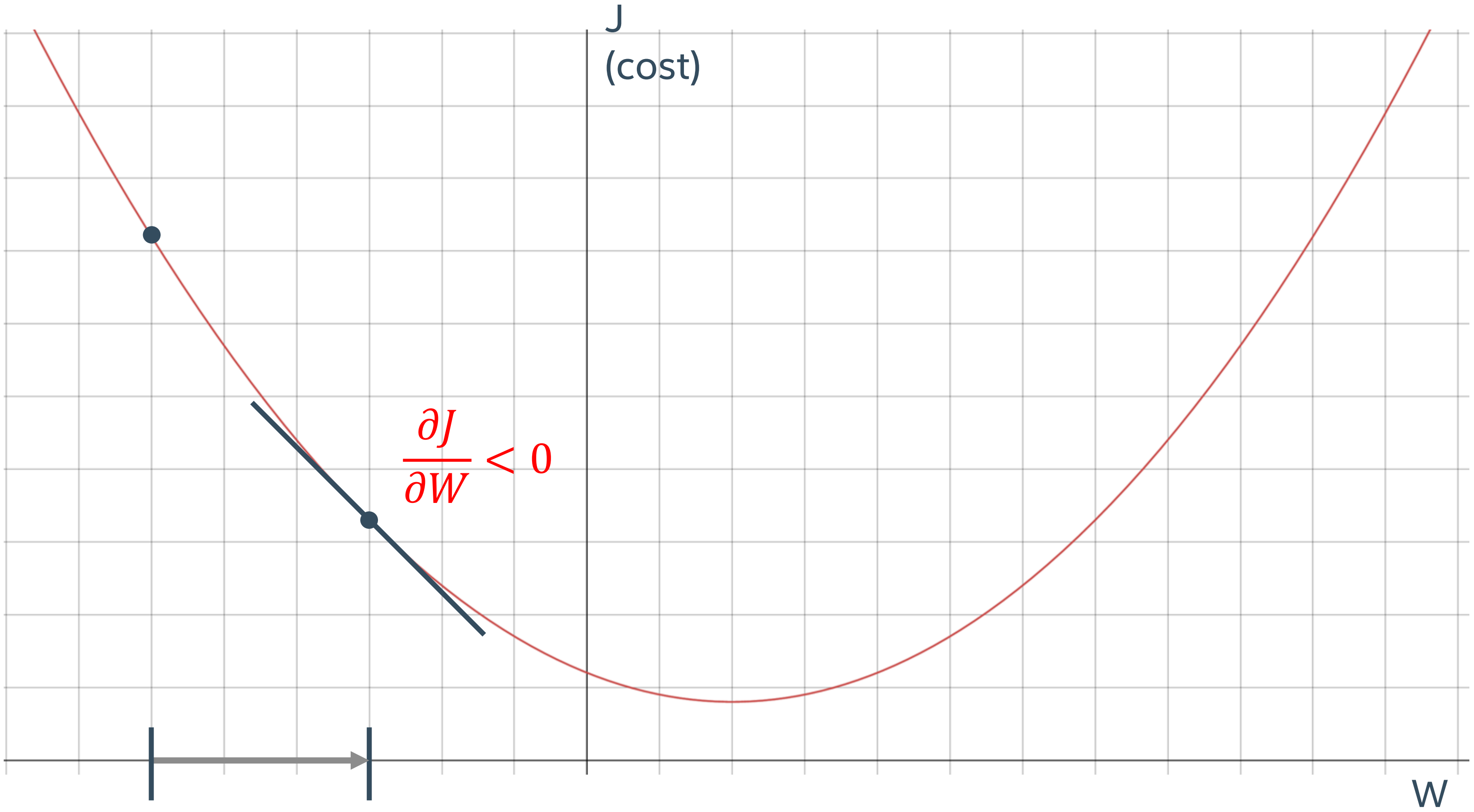

J
(cost)
$\frac{\partial J}{\partial W} < 0$
W

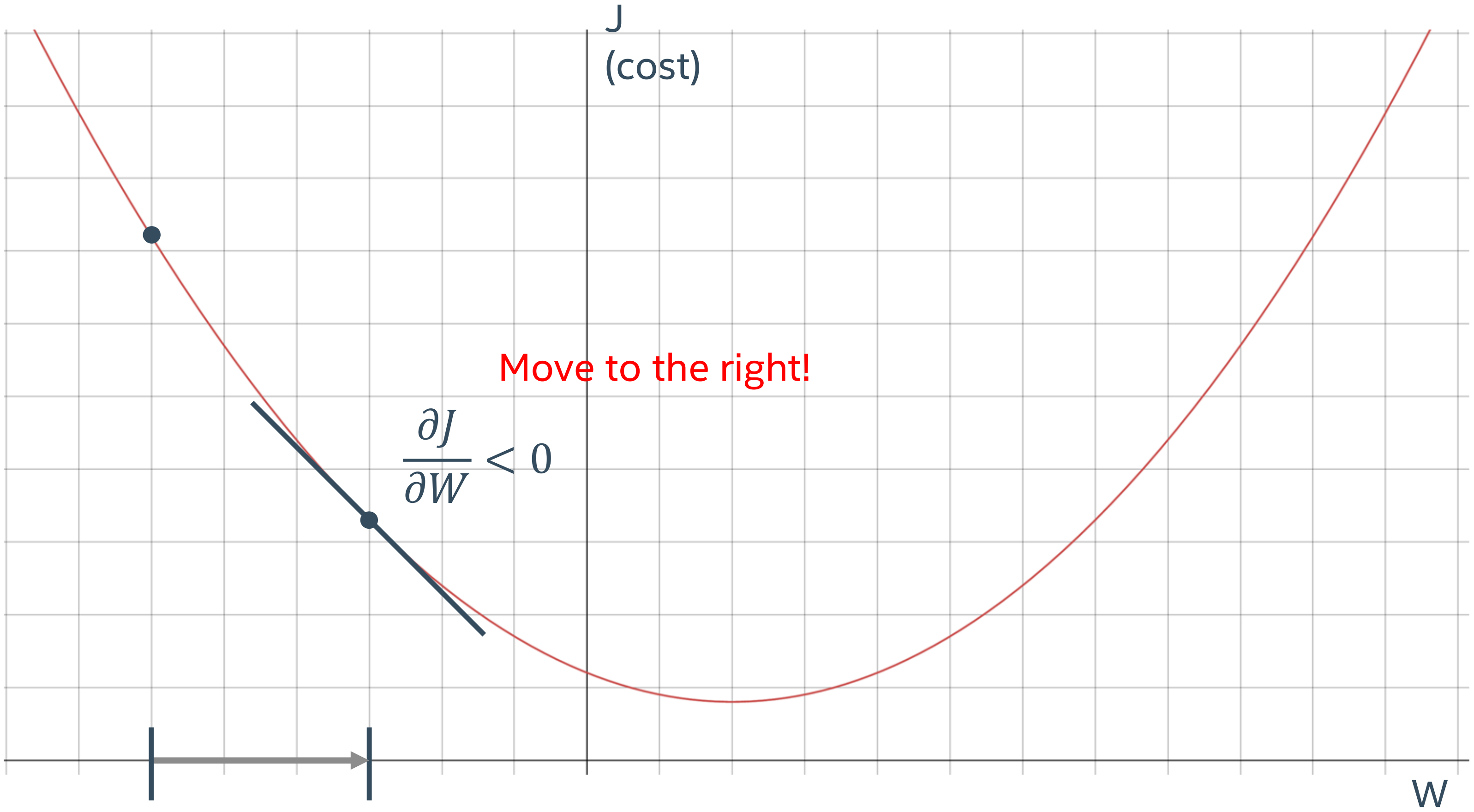

J
(cost)
Move to the right!
$\frac{\partial J}{\partial W} < 0$
W

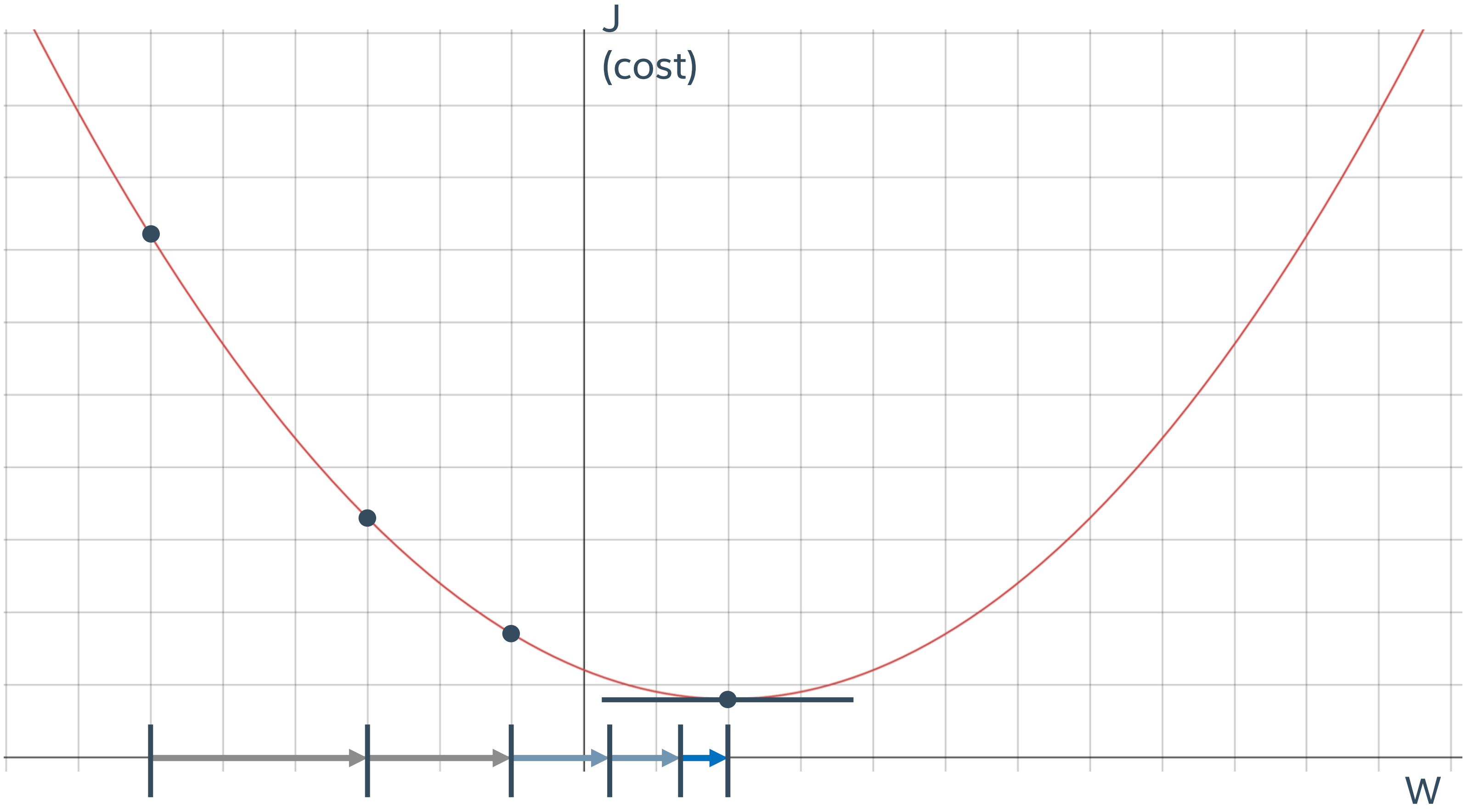

J
(cost)
Move to the right!
$\frac{\partial J}{\partial W} < 0$
ΔW
W


J
(cost)
Cost at new weight value
W


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)
W

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

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

Derivative of MSE

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

THE PROCESS OF DOING GRADIENT DESCENT (MATH-VERSION)

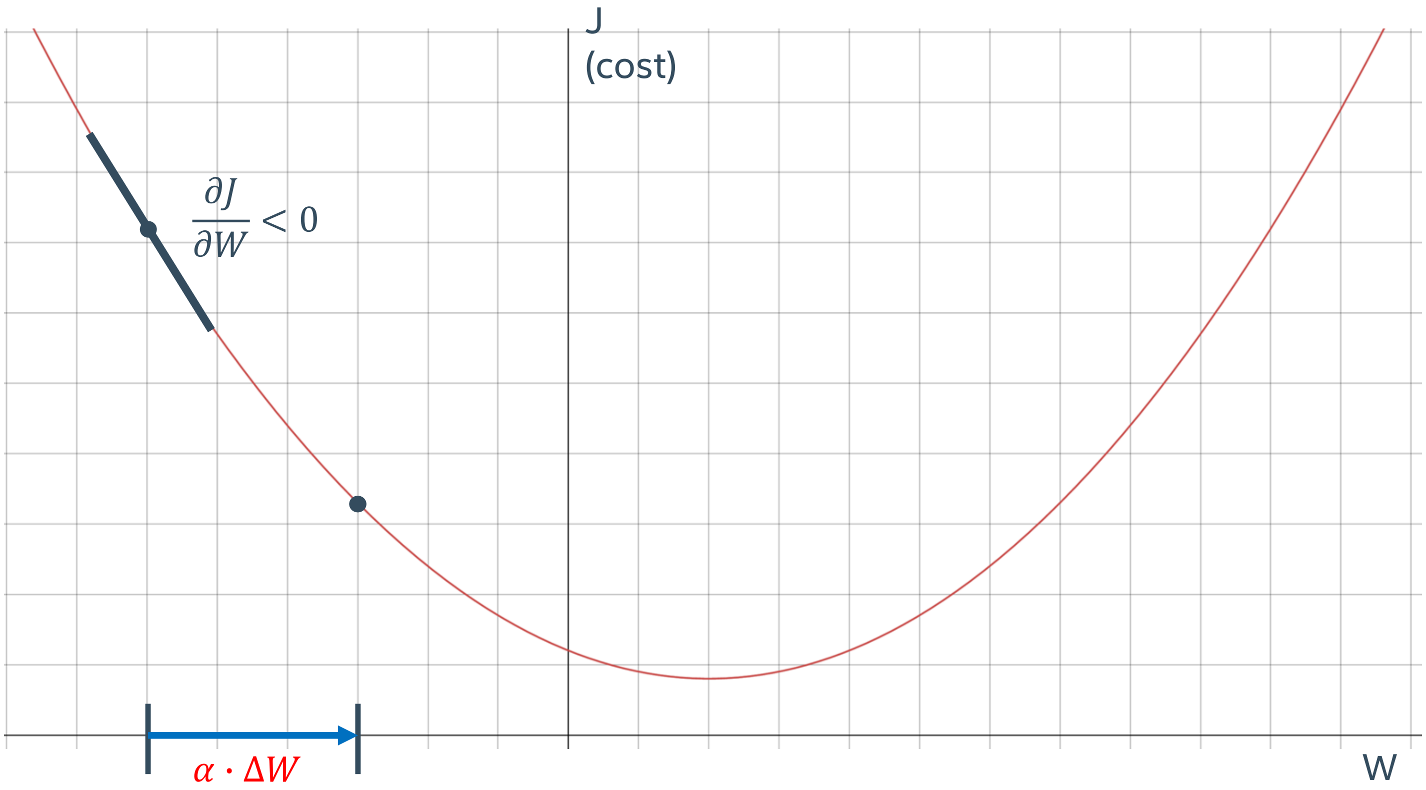
2. **Adjust the weight by subtracting some amount of ΔW**
   - α (alpha) is known as the learning rate
   - It's the first hyper-parameter we've seen in the class

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

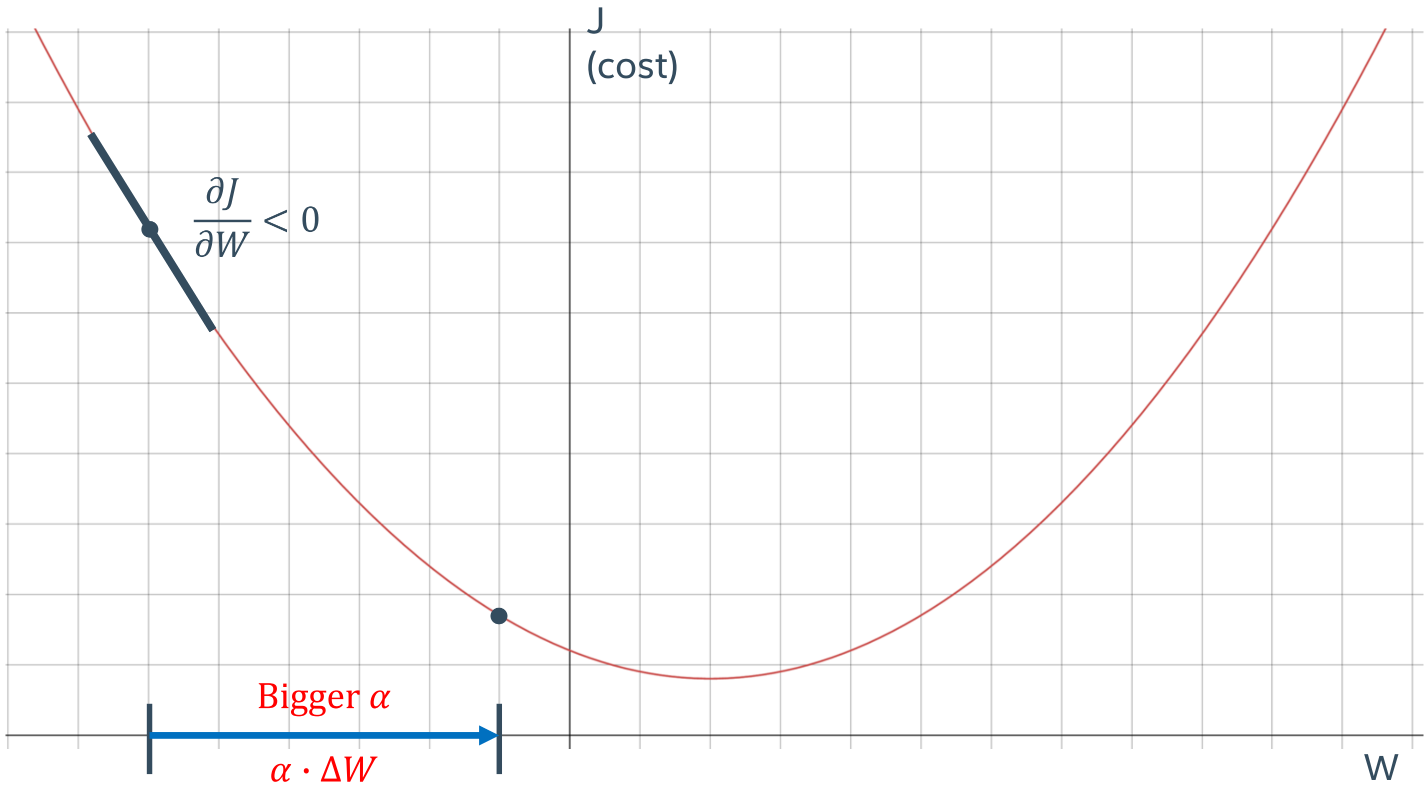
Minus adjusts W in the correct direction

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

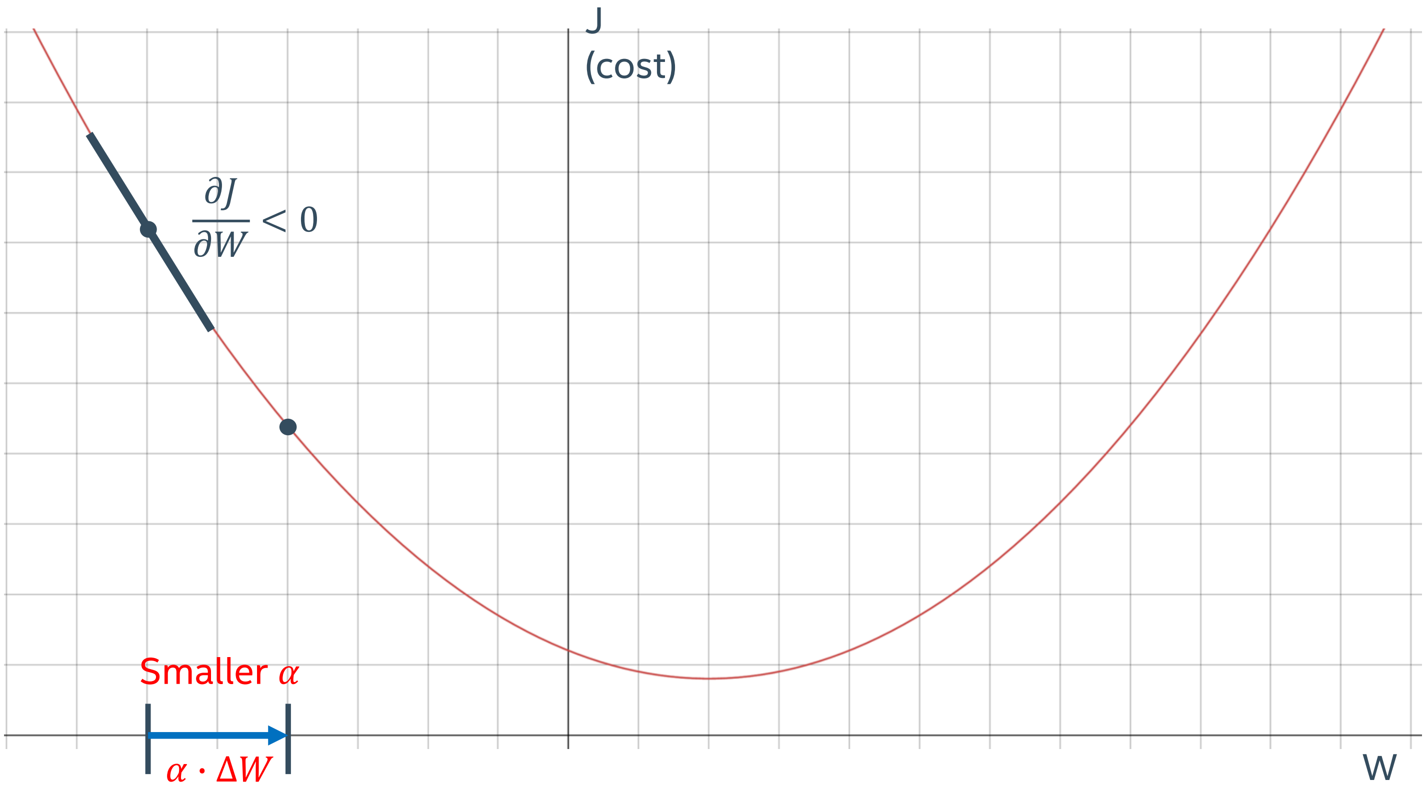
ADJUSTING THE LEARNING RATE



ADJUSTING THE LEARNING RATE

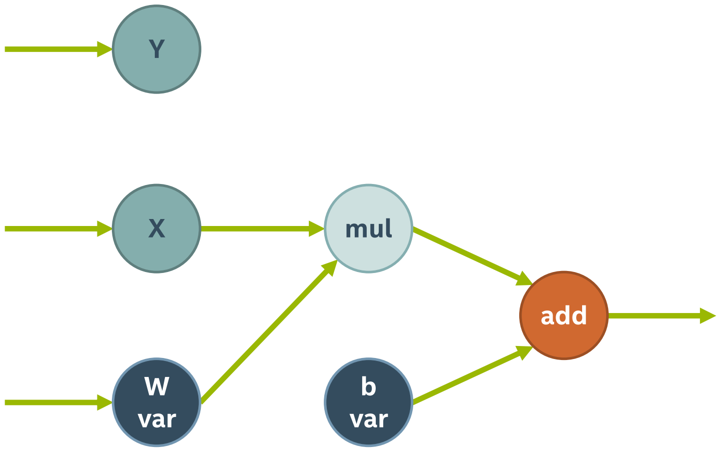


ADJUSTING THE LEARNING RATE



PROCESS OF GRADIENT DECENT (COMPUTATION GRAPH)

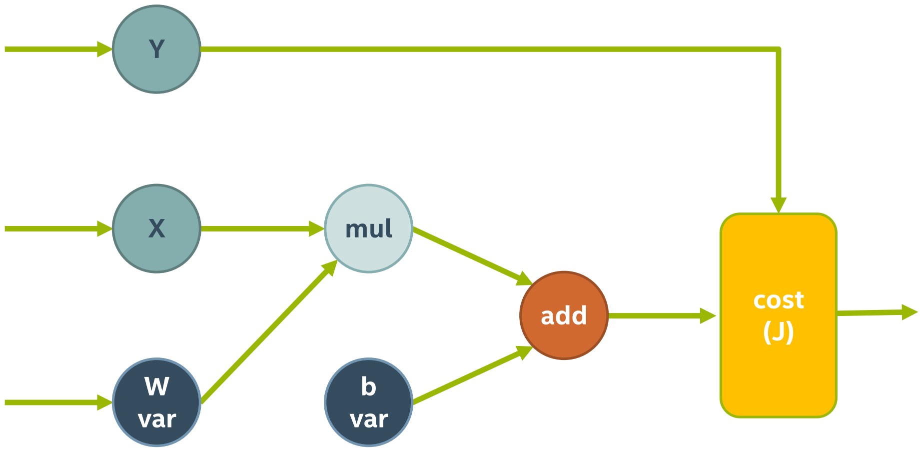
1. **Start with our basic model**
   - $Y = Wx + b$



PROCESS OF GRADIENT DECENT (COMPUTATION GRAPH)

2. Define our cost function

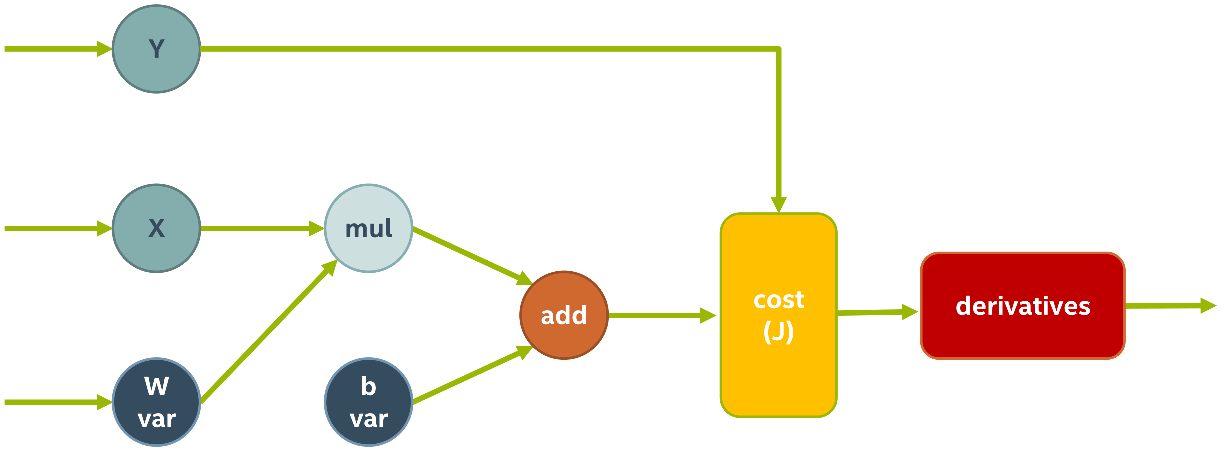
- In this case `tf.square(add - y)`
- Built in: `tf.squared_difference(add, y)`



PROCESS OF GRADIENT DECENT (COMPUTATION GRAPH)

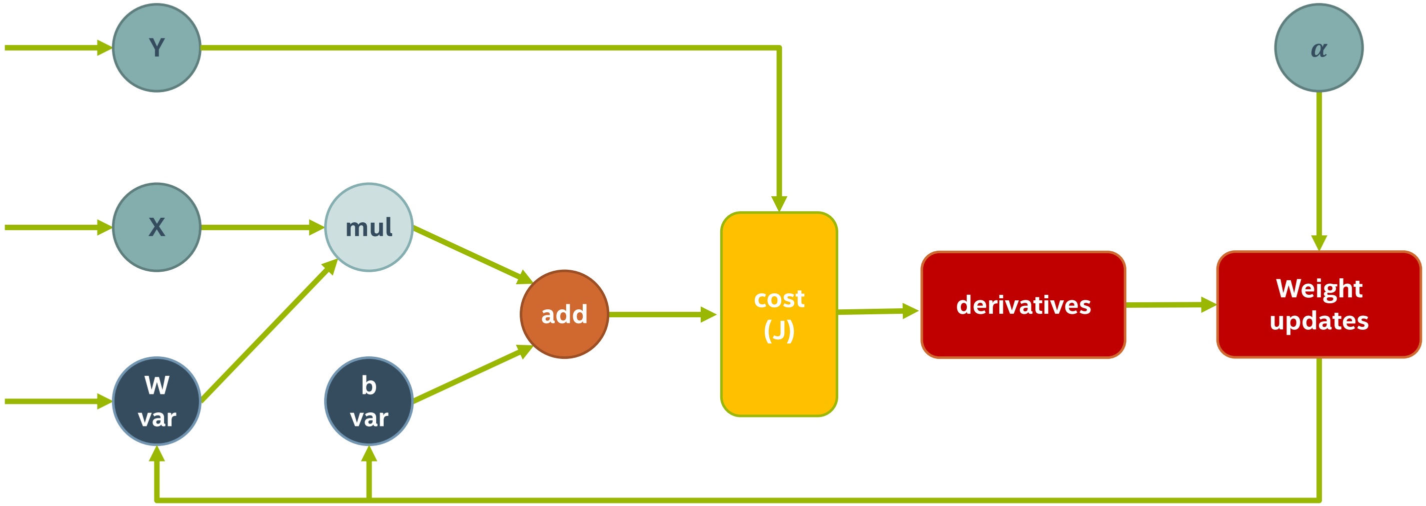
3. Get the derivatives of the cost w.r.t Variables

- $\frac{\partial J}{\partial W}, \frac{\partial J}{\partial b}$
- Sum over all examples = ΔW, Δb



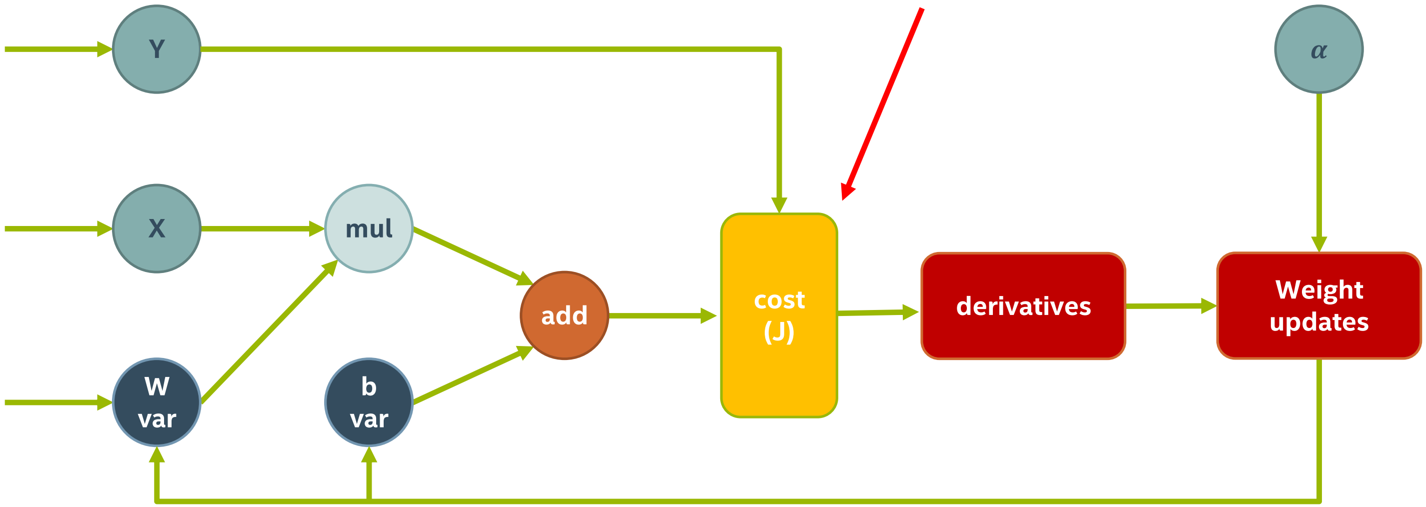
PROCESS OF DOING GRADIENT DECENT (COMPUTATION GRAPH)

4. Use derivatives and learning rate to update Variables
   - $W := W - \alpha \cdot \Delta W$
   - $b := b - \alpha \cdot \Delta b$



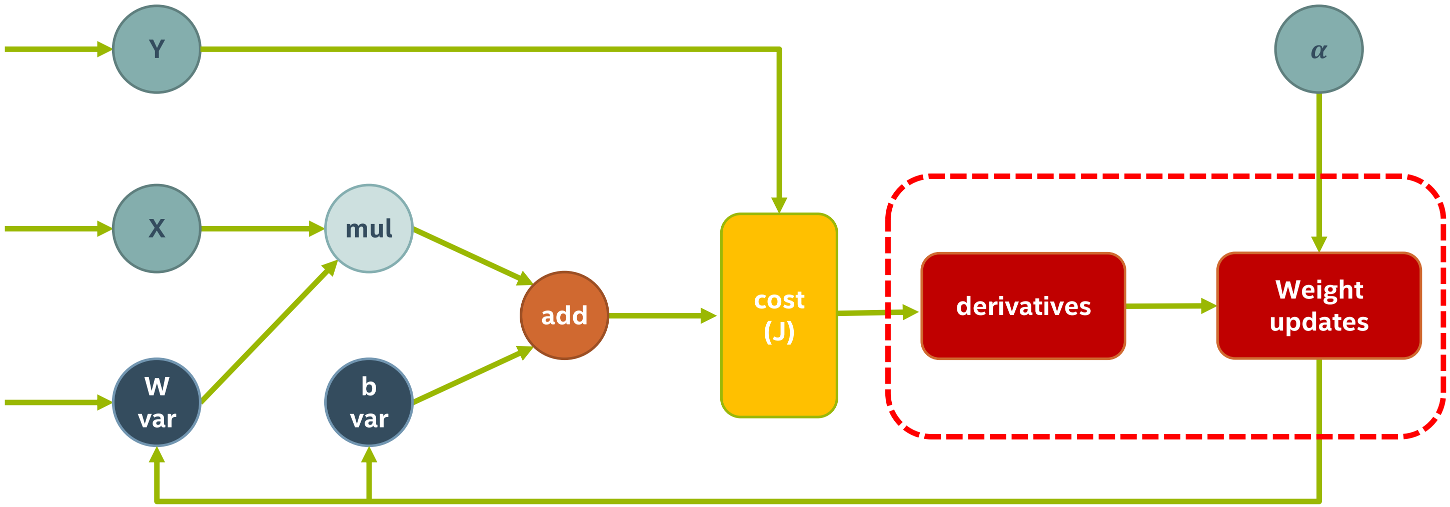
COST NEEDS TO BE DEFINED LIKE OTHER OPERATIONS IN TF

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



DERIVATIVES AND UPDATES USE A TF.OPTIMIZER

`Optimizer` computes derivatives and applies them to Variables



THE OPTIMIZER SUPER-CLASS

Two building block methods:

1. **compute_gradients()**
   - Given a loss and list of Variables, will compute partial derivatives

```
opt = tf.train.GradientDescentOptimizer(learning_rate)
grads = opt.compute_gradients(loss, [W, b])
```

   - You can then perform additional tweaks, if you'd like (not today)

2. **apply_gradients()**
   - Creates an Operation that updates the variables, given gradients

```
train = opt.apply_gradients(grads)
…
sess.run(train, feed_dict)
```

HELPER FUNCTION: OPTIMIZER.MINIMIZE()

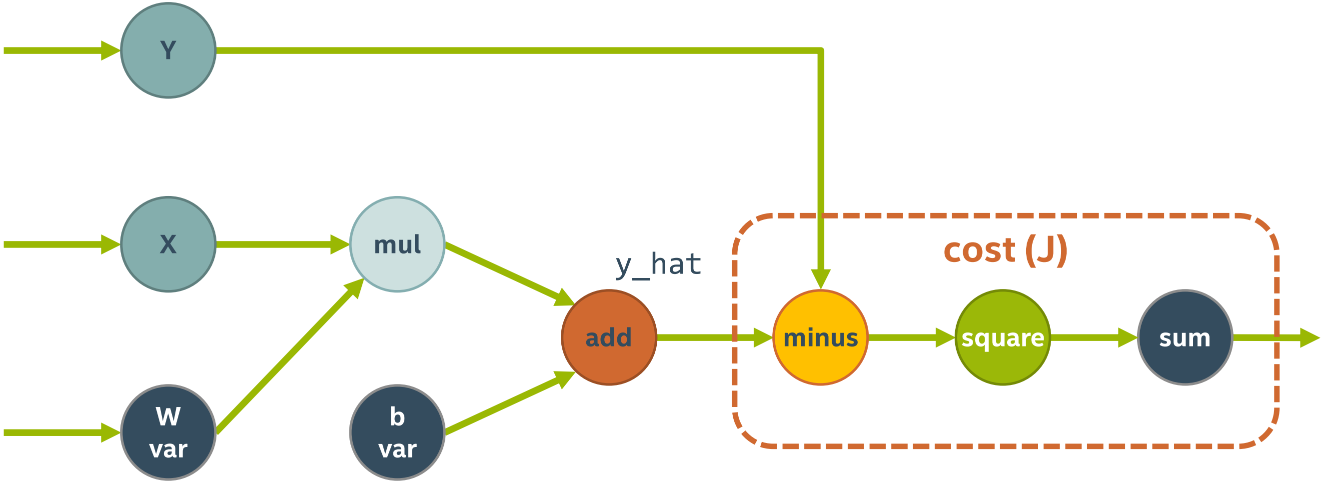
opt.minimize()

- Given a loss target, creates an Operation that automatically computes and applies gradients for all trainable Variables that affect the loss
- Same as using **`compute_gradients and apply_gradients`** without adjusting the gradient values
- Easiest—use this unless you are manually adjusting gradients

```
train = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
…
sess.run(train, feed_dict)
```

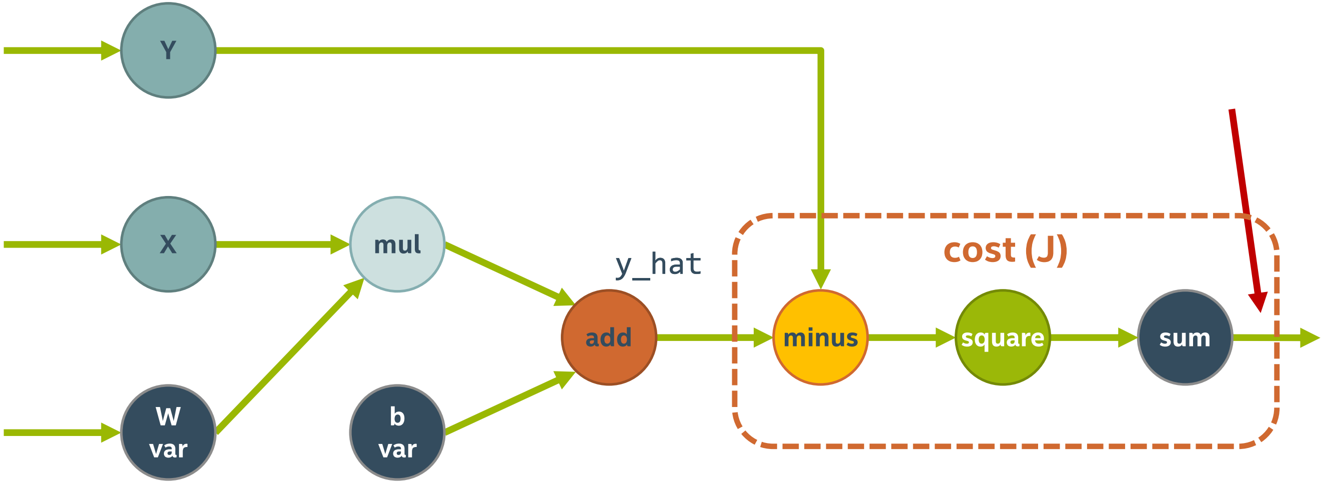
HOW DOES TF.OPTIMIZER WORK?

Let's look at the model up through the loss function



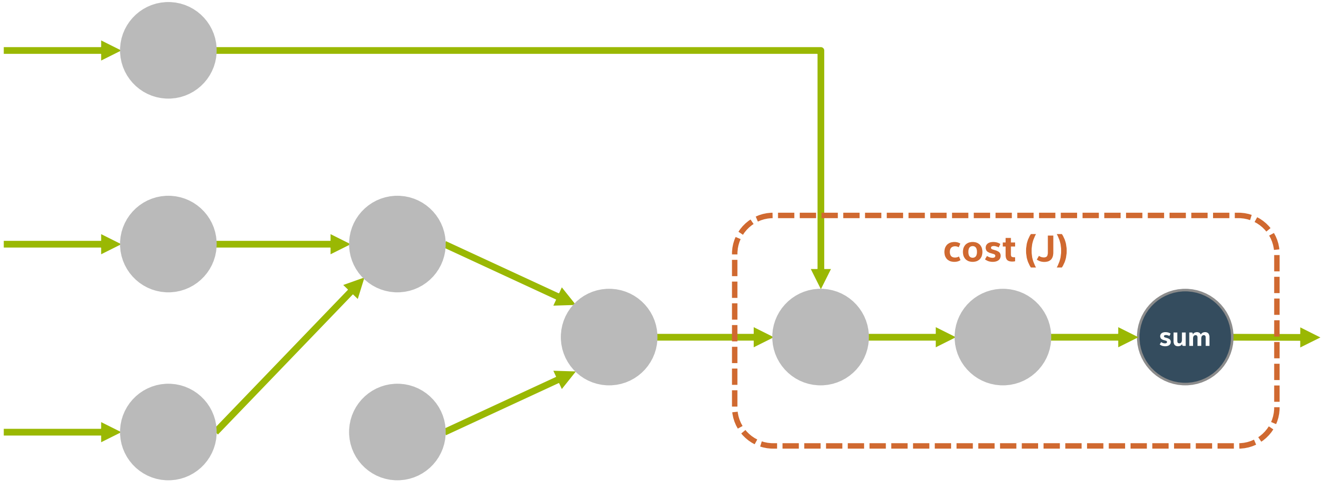
HOW DOES TF.OPTIMIZER WORK?

The output of the sum Operation is our cost, J



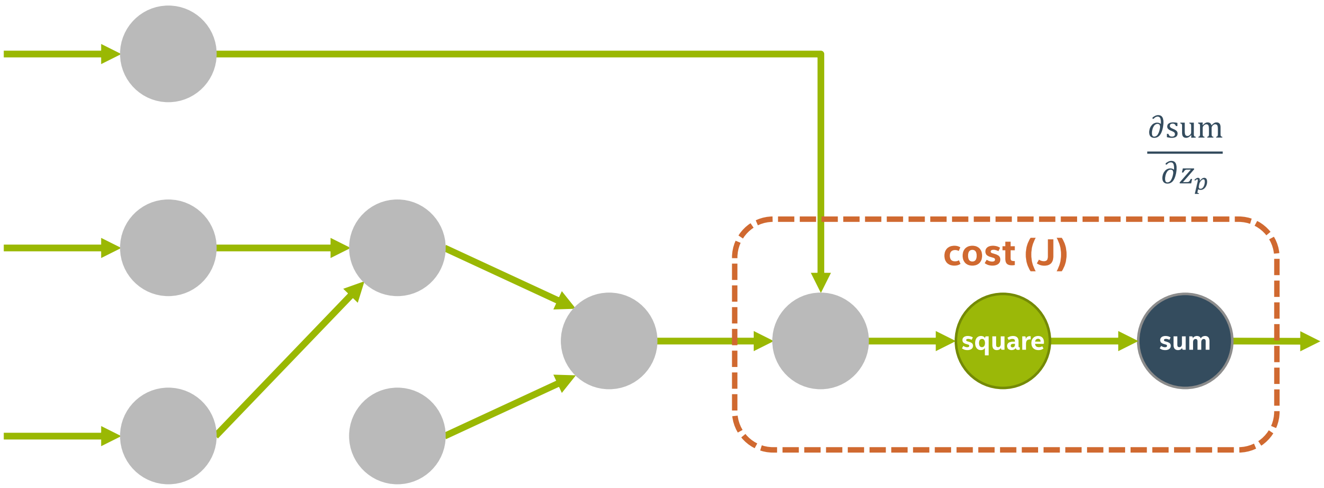
WHAT IS DERIVATIVE OF SUM FUNCTION W.R.T ANY INPUT?

$$\text{sum}(z_1, \ldots, z_n) = \sum_{i=1}^{n} z_i \qquad \frac{\partial sum}{\partial z_p} = 1$$



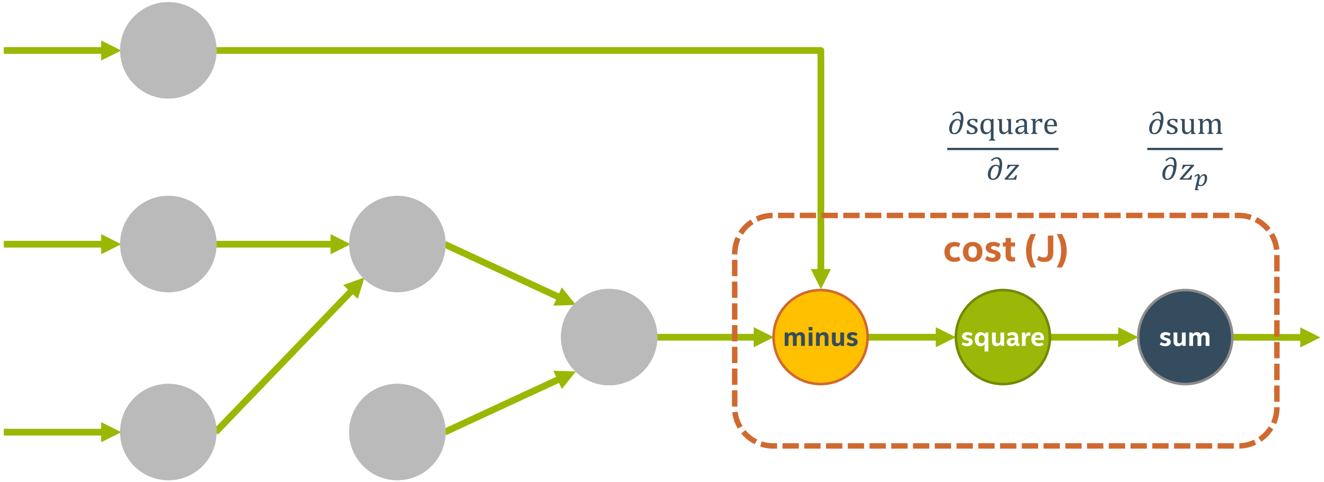
WHAT IS DERIVATIVE OF SQUARE FUNCTION W.R.T ANY INPUT?

$$\text{square}(z) = z^2 \qquad \frac{\partial \text{square}}{\partial z} = 2z$$



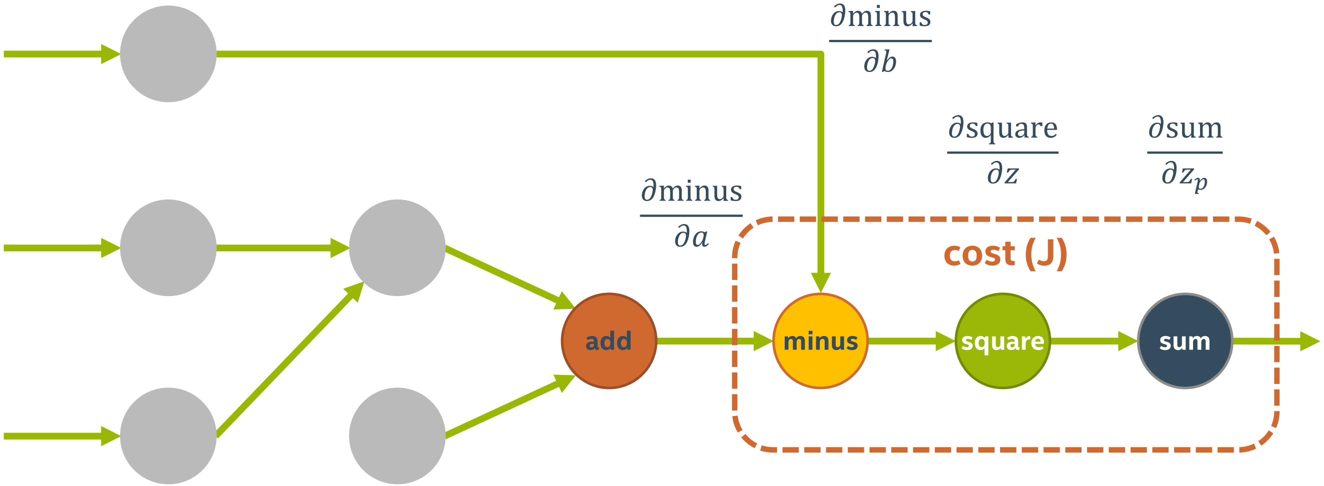
WHAT IS DERIVATIVE OF MINUS W.R.T EITHER INPUT?

$$\text{minus}(a, b) = a - b \qquad \frac{\partial \text{minus}}{\partial a} = 1 \qquad \frac{\partial \text{minus}}{\partial b} = -1$$



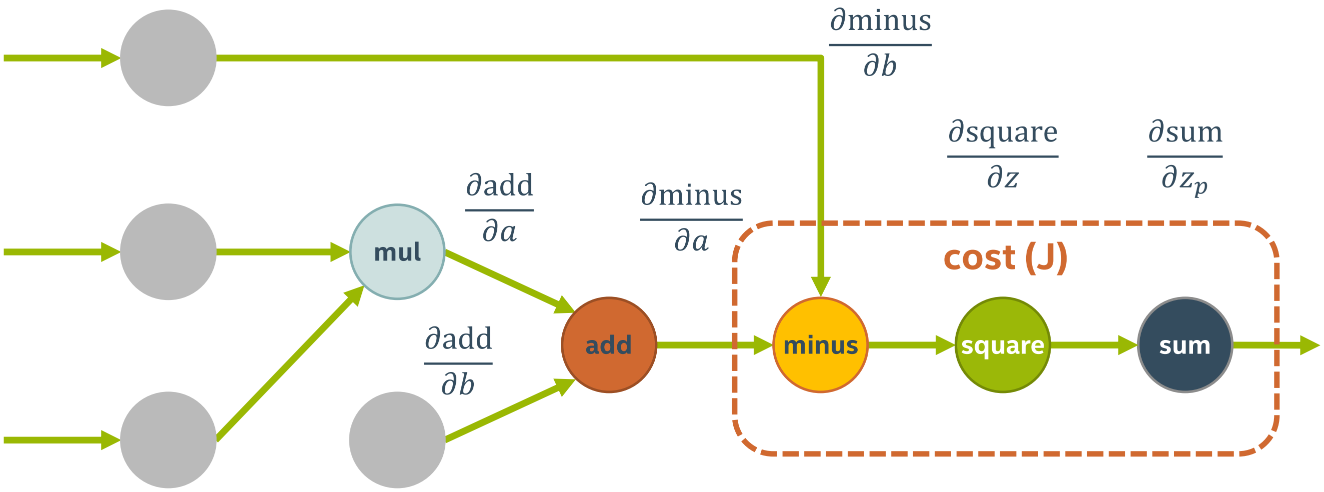
WHAT IS DERIVATIVE OF ADD W.R.T EITHER INPUT?

$$\mathrm{add}(a, b) = a + b \qquad \frac{\partial \mathrm{add}}{\partial a} = 1 \qquad \frac{\partial \mathrm{add}}{\partial b} = 1$$



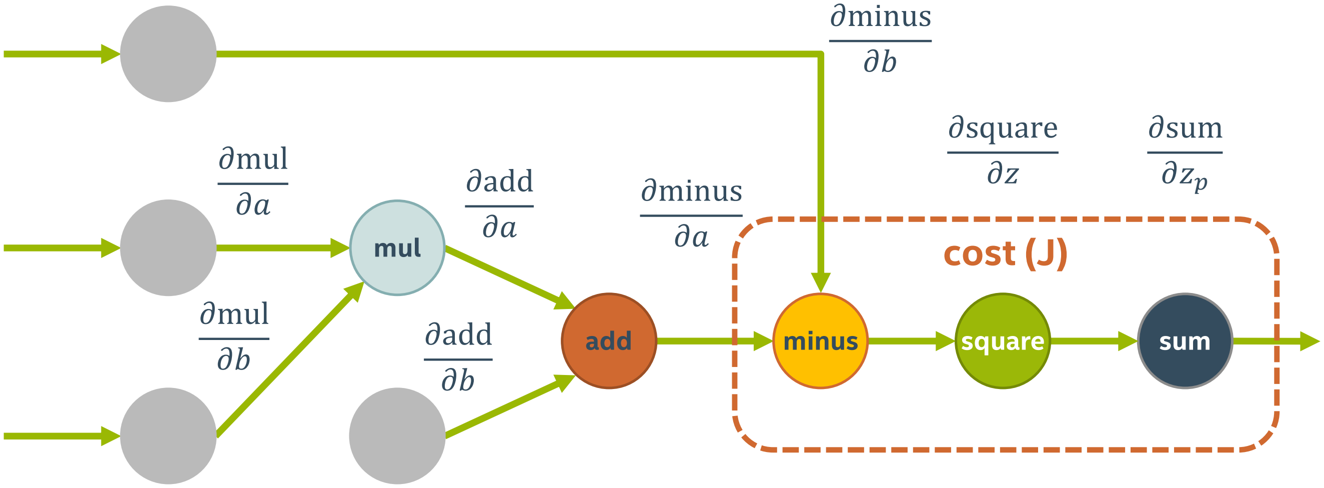
WHAT IS DERIVATIVE OF MULTIPLICATION W.R.T EITHER INPUT?

$$\mathrm{mul}(a, b) = a \times b \qquad \frac{\partial \mathrm{mul}}{\partial a} = b \qquad \frac{\partial \mathrm{mul}}{\partial b} = a$$



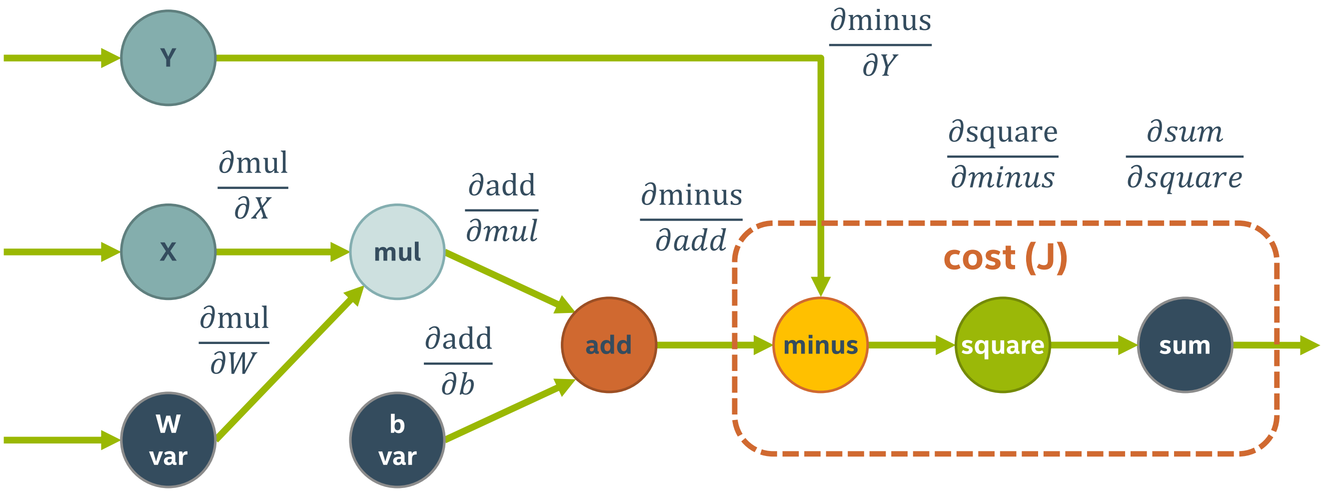
WHAT IS DERIVATIVE OF MULTIPLICATION W.R.T EITHER INPUT?

$$\mathrm{mul}(a,b) = a \times b \qquad \frac{\partial \mathrm{mul}}{\partial a} = b \qquad \frac{\partial \mathrm{mul}}{\partial b} = a$$

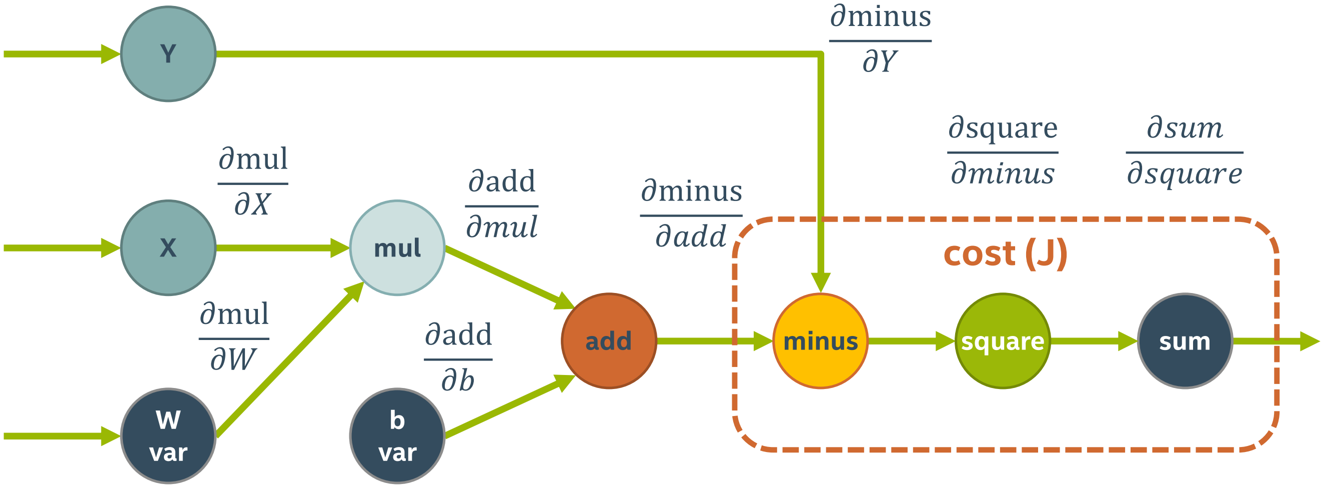


NOW, WE MOVE FROM GENERIC ARGUMENTS TO SPECIFIC LINKS (OUTPUTS) IN OUR GRAPH.

$$\frac{\partial sum}{\partial z_p} \rightarrow \frac{\partial sum}{\partial square} \quad \frac{\partial square}{\partial z} \rightarrow \frac{\partial square}{\partial minus}$$

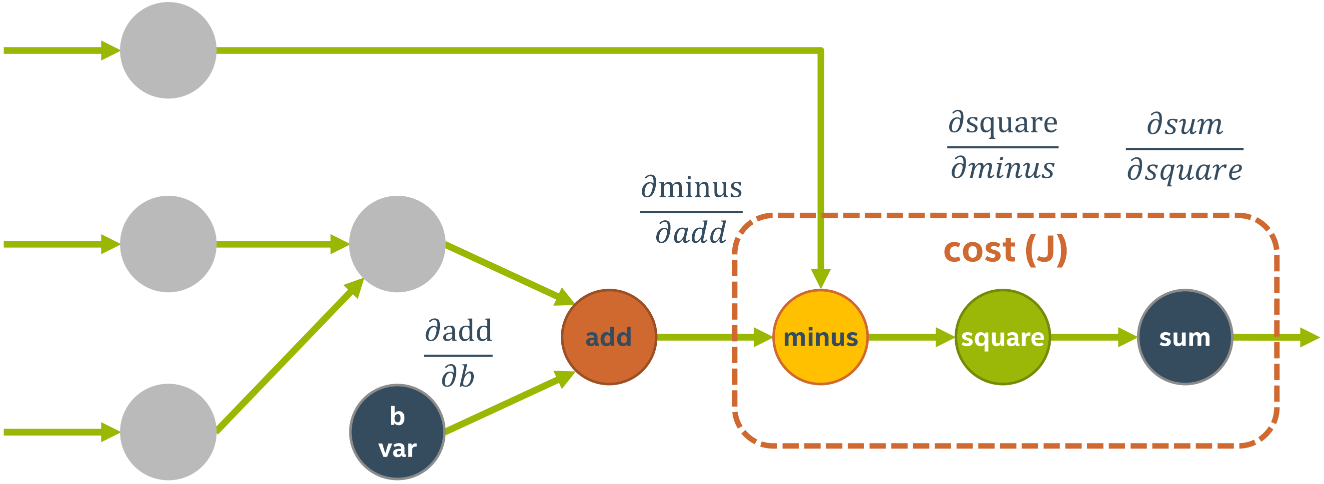


NOW WE CAN USE THE CHAIN RULE TO GET DERIVATIVE W.R.T WEIGHTS!



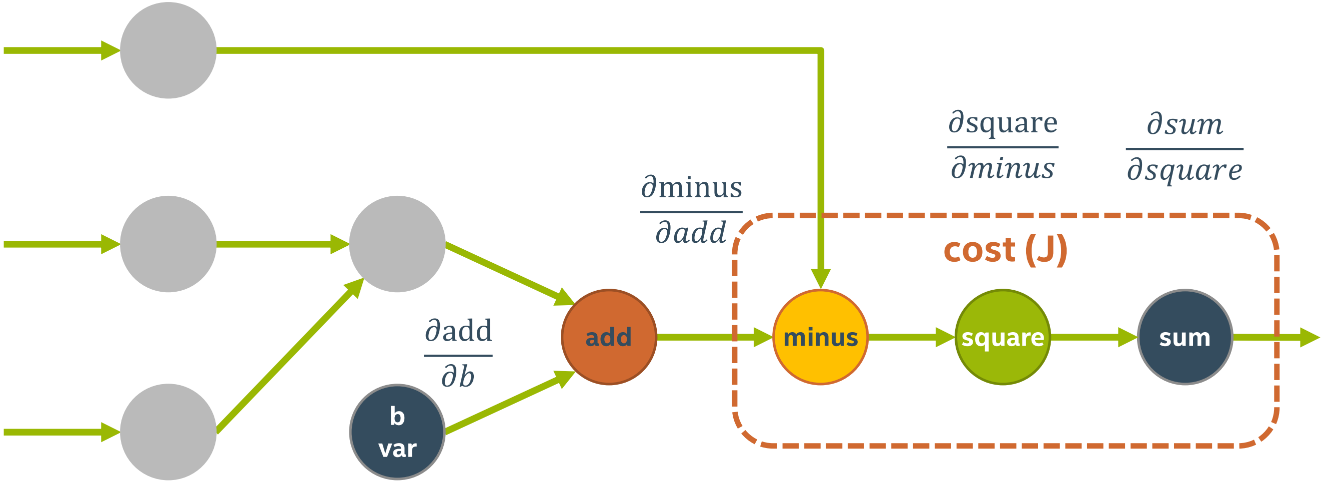
WITH RESPECT TO B:

$$J = sum; \quad \frac{\partial sum}{\partial b} = \frac{\partial sum}{\partial square} \cdot \frac{\partial square}{\partial minus} \cdot \frac{\partial minus}{\partial add} \cdot \frac{\partial add}{\partial b}$$



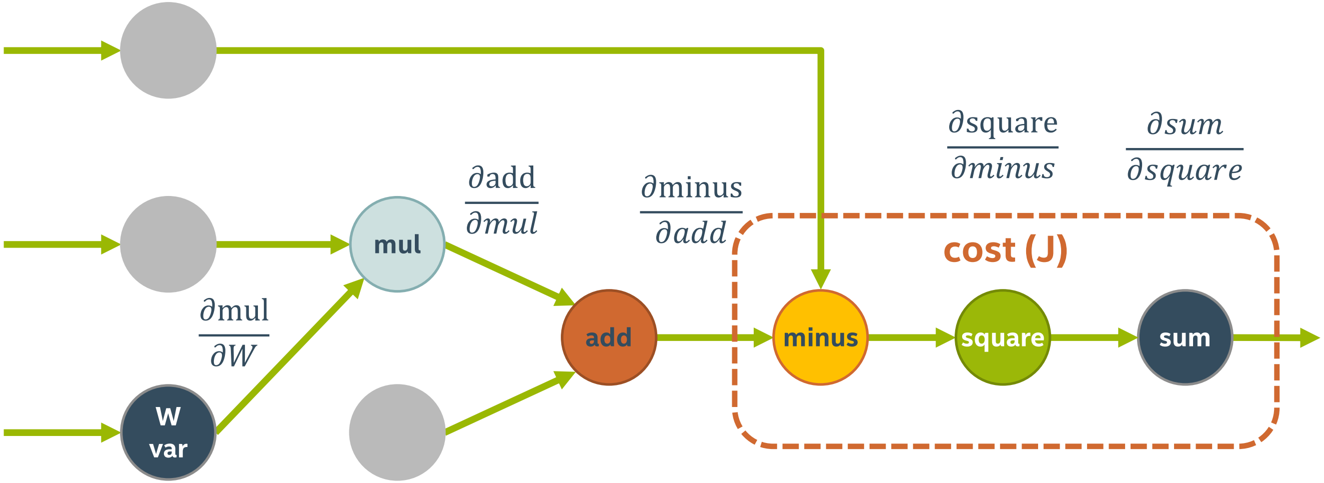
REMEMBER THE "CANCELLING" EFFECT OF CHAIN RULE:

$$J = sum; \quad \frac{\partial sum}{\partial b} = \frac{\partial sum}{\cancel{\partial square}} \cdot \frac{\cancel{\partial square}}{\cancel{\partial minus}} \cdot \frac{\cancel{\partial minus}}{\cancel{\partial add}} \cdot \frac{\cancel{\partial add}}{\partial b}$$



SIMILAR PROCESS FOR DERIVATIVE WITH RESPECT TO W

$$J = sum; \quad \frac{\partial sum}{\partial W} = \frac{\partial sum}{\partial square} \cdot \frac{\partial square}{\partial minus} \cdot \frac{\partial minus}{\partial add} \cdot \frac{\partial add}{\partial mul} \cdot \frac{\partial mul}{\partial W}$$



THIS IS AUTOMATIC DIFFERENTIATION

By using easily differentiable pieces, we get the derivative with respect to arbitrary inputs

Generalizes as long as you use differentiable pieces

This is how TensorFlow, Theano, Torch, etc. work

- Each differentiable Operation has a “gradient” function that defines how to take the derivative w.r.t its different inputs

This is essentially Backpropagation

VECTORIZATION

PARALLELIZING OUR MATH

So far, we've limited our model:

- Single-variable regression
  - Or, at least explicit placeholders for each input x
- Only processing one example at a time

We can do better on both:

- Multi-variable regression - with only one placeholder input!
- Process many pieces of data in a "batch"

VECTORIZATION #1: MULTI-VARIABLE REGRESSION

Let's modify the function for our linear model:

$$\hat{Y} = X^T W + b$$

$\hat{Y} \in \mathbb{R}$ - an scalar prediction

$X \in \mathbb{R}^m$ - an m length vector. Inputs $x_1, x_2, \dots, x_m$

$W \in \mathbb{R}^m$ - an m-dimensional vector.

- m weights corresponding to $x_1, x_2, \dots, x_m$

$b \in \mathbb{R}$ - a scalar bias

VECTORIZATION #1: MULTI-VARIABLE REGRESSION

Double check that matrix dimensions work out

$$\hat{Y} = X^T W + b$$

[1]   [1,m] x [m,1]   [1]

[1]

HOW TO ACCOMPLISH THIS IN TENSORFLOW

Give x placeholder and W weight a vector shape:

```
x = tf.placeholder(tf.float32, shape=[m, 1])
W = tf.Variable(tf.truncated_normal([m,1]))
```

Use matrix multiplication and transpose to calculate

```
y_hat = tf.matmul(tf.transpose(x), W) + b
```

VECTORIZATION #2: BATCH EXAMPLES

Let's modify the function for our linear model:

$$\hat{Y} = X^T W + b$$

$\hat{Y} \in \mathbb{R}^n$ - an n-dimensional vector.

- Predictions for n examples

$X \in \mathbb{R}^{n \times m}$ - an n-by-m matrix.

- Inputs $x_1, x_2, \ldots, x_m$ for n examples

$W \in \mathbb{R}^m$ - an m-dimensional vector.

- m weights corresponding to $x_1, x_2, \ldots, x_m$

$b \in \mathbb{R}^n$ - a vector of identical bias numbers

VECTORIZATION #2: BATCH EXAMPLES

Double check that matrix dimensions work out

$$\hat{Y} = X \quad W + b$$

[n,1] [n,m] x [m,1] [n,1]

[n,1]

HOW TO ACCOMPLISH THIS IN TENSORFLOW

Give x placeholder matrix shape, W vector shape:

```
x = tf.placeholder(tf.float32, shape=[None, m])
W = tf.Variable(tf.truncated_normal([m,1]))
```

- None in a TensorFlow shape means that any length is allowed
- Allows you to input any amount of training examples

Use matrix multiplication to calculate

```
y_hat = tf.matmul(x, W) + b
```

This is what we'll end up doing for most of this class.

ODDS AND ENDS

TRAINING, VALIDATION, AND TEST SETS

In order to better evaluate our model, we split our data into **training**, **validation**, and **test** sets

Training data is used to train the model as we discussed

Validation data is used to periodically evaluate the model on held-out data

Test data is only ever used once, as a final evaluation of the model

Common split percentages are 60%, 20%, 20%

TRAINING, VALIDATION, AND TEST SETS

We do this to evaluate how much our model is **overfitting** the data, or "memorizing" answers

We want the model to generalize to examples not in the training data

LOGISTIC REGRESSION (BINARY CLASSIFICATION)

$$J = -\frac{1}{n}\sum_{i=1}^{n} y_i(\log(\hat{y}_i)) + (1 - y_i)(\log(1 - \hat{y}_i))$$

$$\hat{y}_i = \frac{1}{1 + e^{-z_i}}$$

$$z_i = W^{\mathrm{T}} X_i + b$$

BUT WAIT, THERE'S MORE!

Regularization

Input normalization

But we'll hold off until next week.

Nervana™ AI Academy