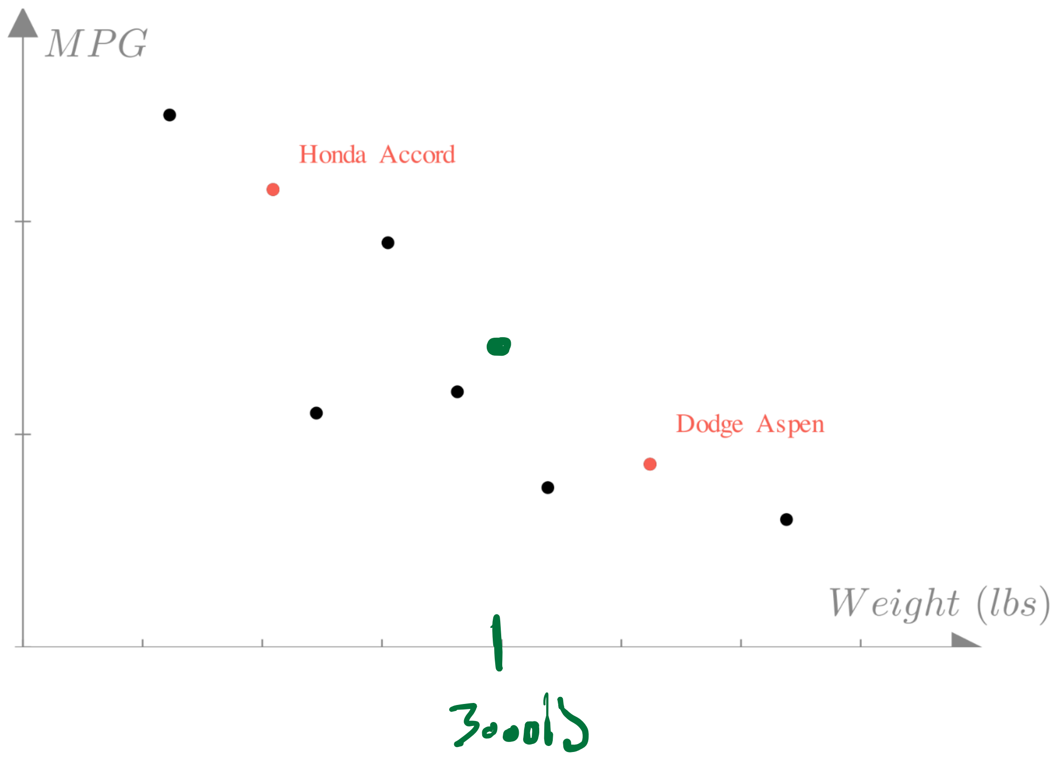
$$\text{Honda Accord: } \begin{bmatrix} \text{Weight:} & \text{2500 lbs} \\ \text{Horsepower:} & \text{123 HP} \\ \text{Displacement:} & \text{2.4 L} \\ \text{0-60mph:} & \text{7.8 Sec} \end{bmatrix} \longrightarrow \text{MPG: 33mpg}$$

$$\text{Dodge Aspen: } \begin{bmatrix} \text{Weight:} & \text{3800 lbs} \\ \text{Horsepower:} & \text{155 HP} \\ \text{Displacement:} & \text{3.2 L} \\ \text{0-60mph:} & \text{6.8 Sec} \end{bmatrix} \longrightarrow \text{MPG: 21mpg}$$

$$\vdots \quad \vdots$$

$$\begin{bmatrix} \text{Weight} \\ \text{hP} \\ \text{Displact} \\ \text{Acc.} \end{bmatrix} \longrightarrow \text{MPG ?}$$



Dataset: $\mathcal{D} = \{(\mathbf{x}_i, y_i) \text{ for } i \in 1 \ldots N\}$

$$\text{Input: } \mathbf{x}_i = \begin{bmatrix} \text{Weight} \\ \text{Horsepower} \\ \text{Displacement} \\ \text{0-60mph} \end{bmatrix}, \quad \text{Output: } y_i = MPG$$

x_i: Vector input

y_i: Scalar output

i: Index of obs

D: Data set

N: size of D

$$\underbrace{\text{mpg} = f(\text{weight, horsepower}\ldots)}_{\text{function}}$$

$$y = f(x)$$

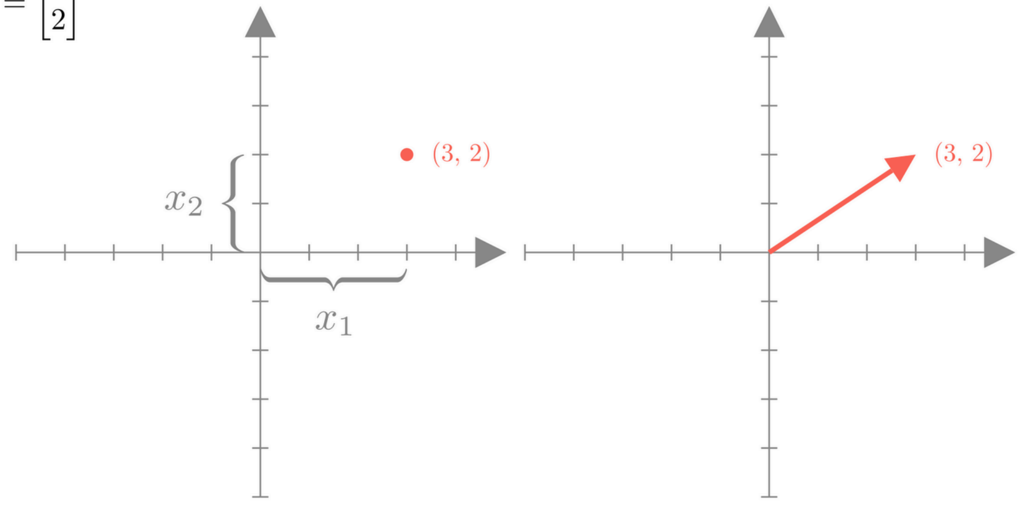
Vectors

$$\underbrace{\mathbf{x}}_{\text{bolded}} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \Big\} d = 3$$

scalar

```
x = np.array([2, 5, 1])
```

$\mathbf{x} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$



x^T

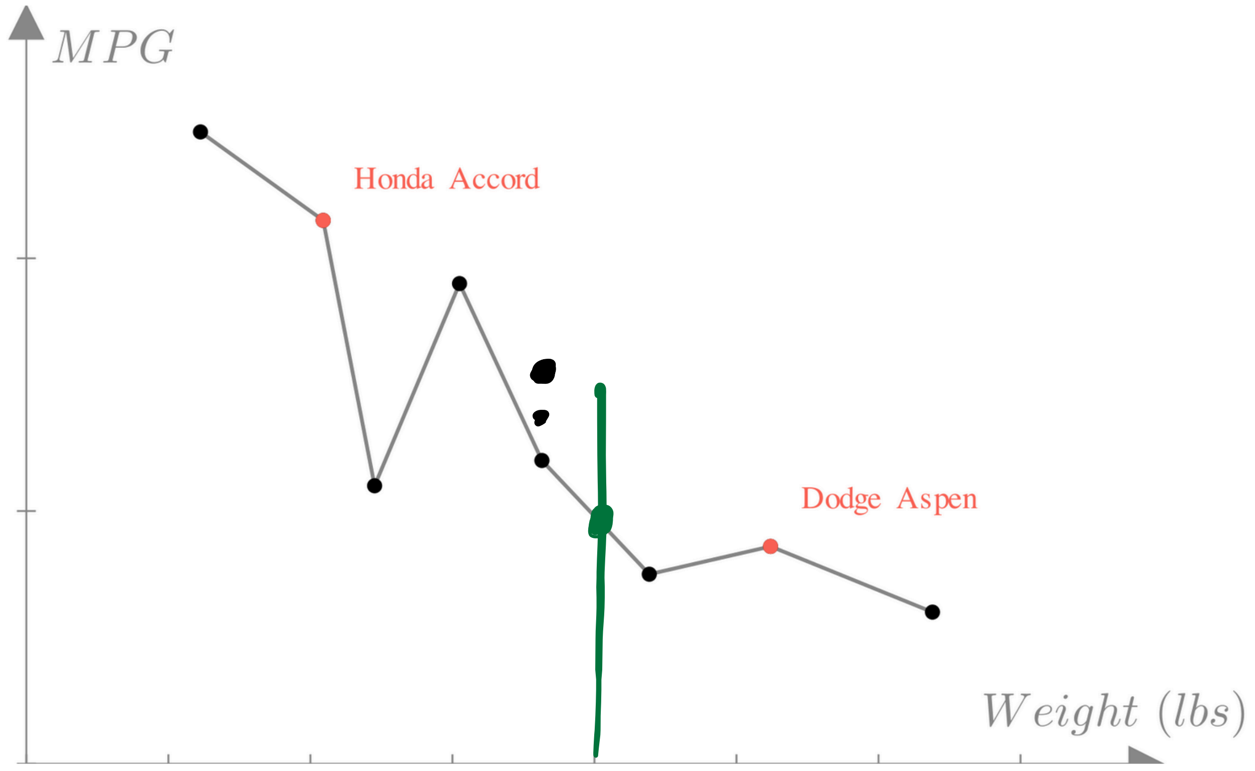
$\downarrow$

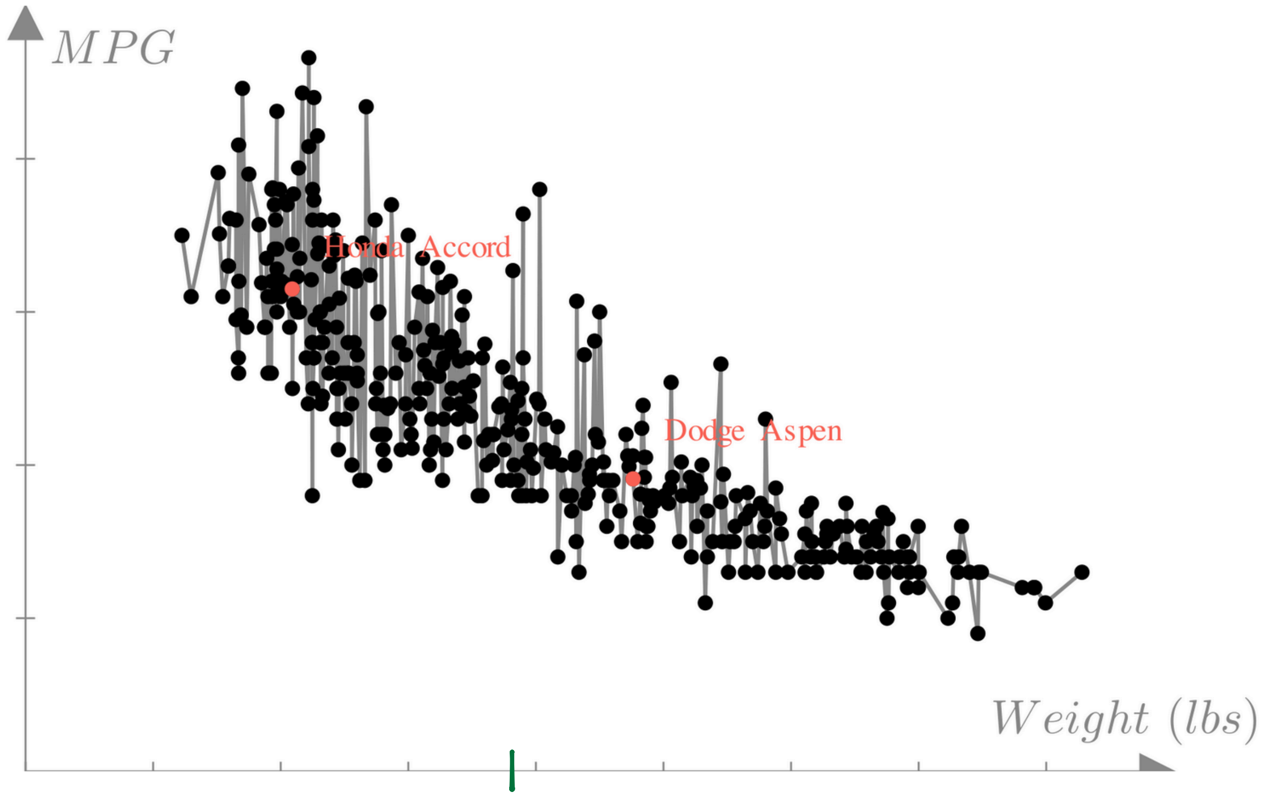
Column vector: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $\underbrace{\text{Row vector: } \mathbf{x}^T = [x_1 \quad x_2 \quad x_3]}$

$|||$ $d \times 1$ matrix

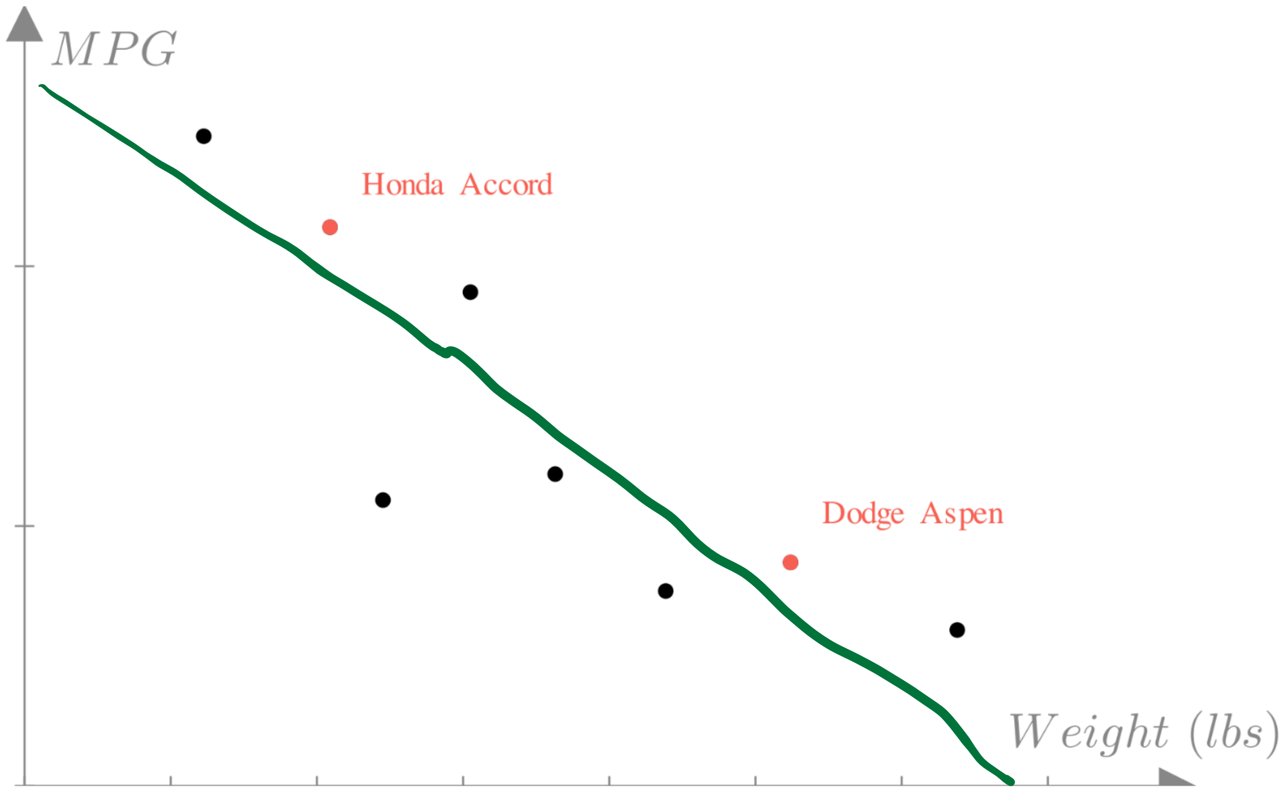
$|||$ $1 \times d$ matrix

linear interpolation




MPG
Honda Accord
Dodge Aspen
Weight (lbs)

Linear Regression



A *linear function* is any function f where the following conditions always hold:

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$

and

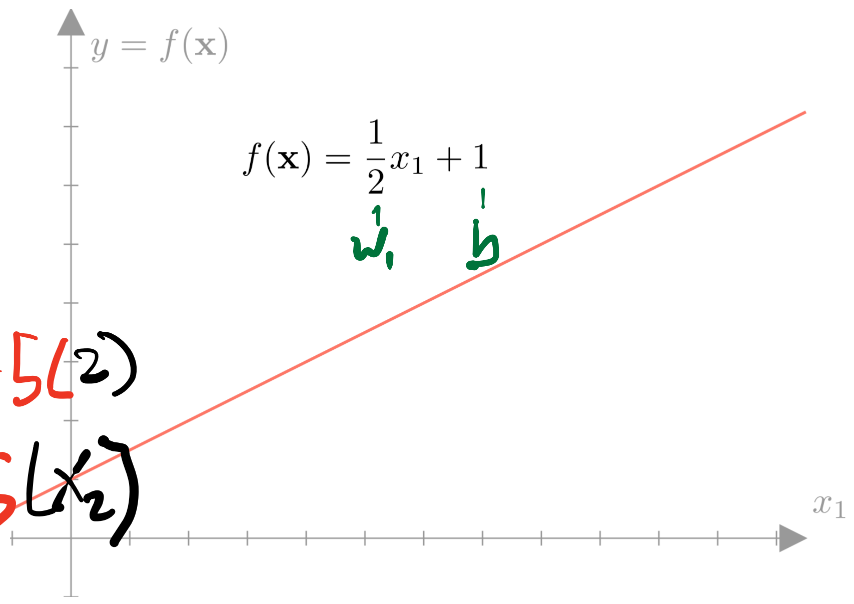
$$f(a\mathbf{x}) = a f(\mathbf{x})$$

For a linear function, the output can be defined as a weighted sum of the inputs. In other words a linear function of a vector can always be written as:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i w_i$$

We can add an offset b to create an *affine* function:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i w_i + b$$



$$f\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = 3(1)+5(2)$$

$$f(x) = 3(x_1)+5(x_2)$$

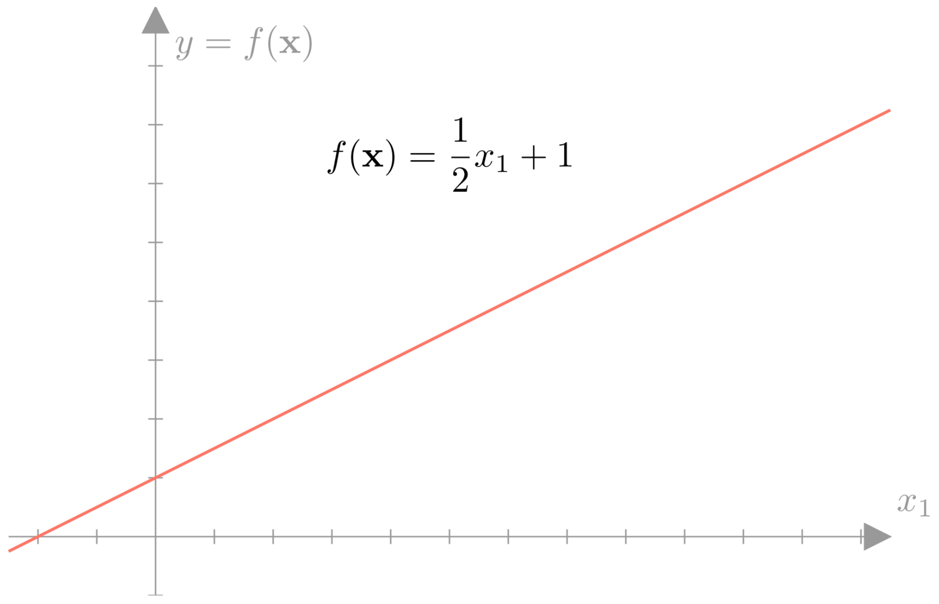
$$x \quad w$$

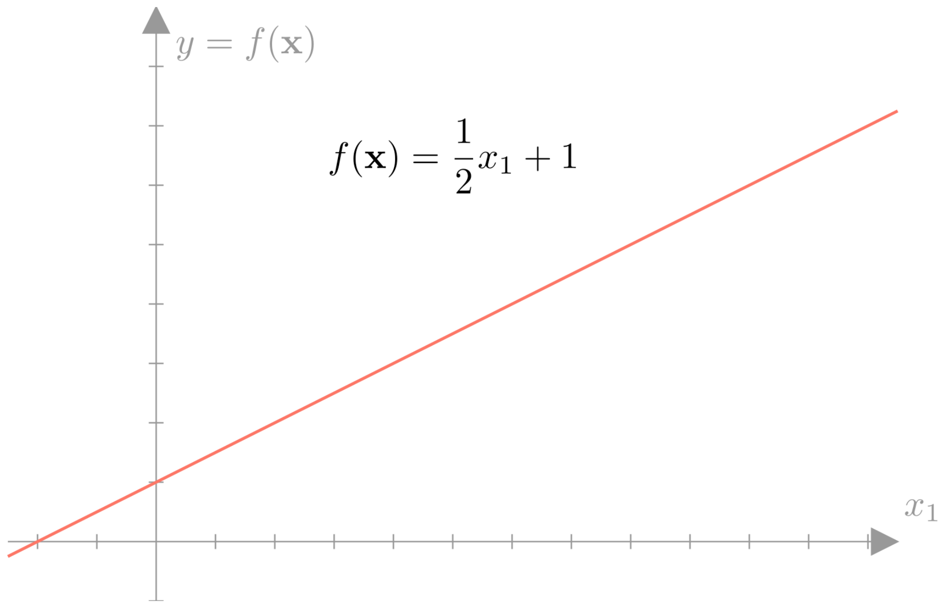
$$f(\bar{x}) = \sum_{i=1}^{n} x_i w_i = x \cdot w = x^T w = [\longrightarrow]\begin{bmatrix}\downarrow\end{bmatrix}$$

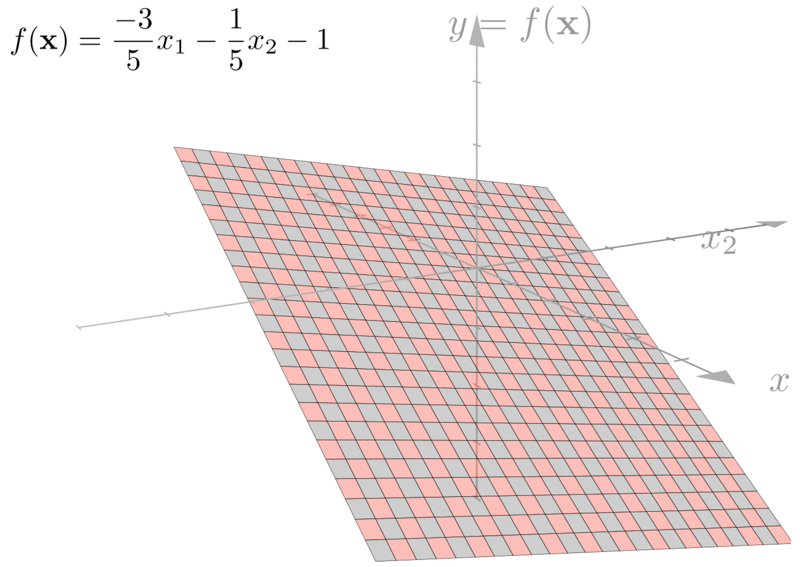
$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} = \sum_{i=1}^{n} x_i w_i$$

```
def f(x):
    w = np.array([-0.6, -0.2])
    b = -1
    return np.dot(x, w) + b
```

Numpy




$y = f(\mathbf{x})$
$f(\mathbf{x}) = \frac{1}{2}x_1 + 1$
x_1


$f(\mathbf{x}) = \frac{-3}{5}x_1 - \frac{1}{5}x_2 - 1$
$y = f(\mathbf{x})$
x_2
x

Linear regression is the approach of modeling an unknown function with a linear function. From our discussion of linear functions, we know that this means that we will make predictions using a function of the form:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} = \sum_{i=1}^{n} x_i w_i$$

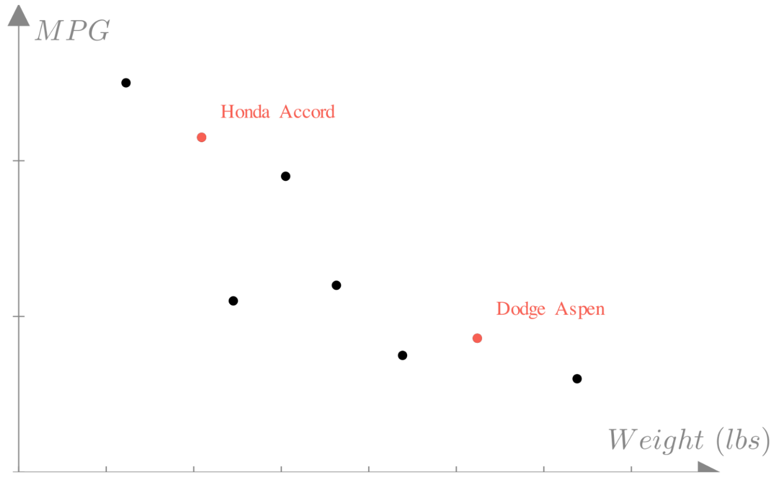
Meaning that the output will be a weighted sum of the *features* of the input. In the case of our car example, we will make predictions as:

$$\text{Predicted MPG} = f(\mathbf{x}) =$$

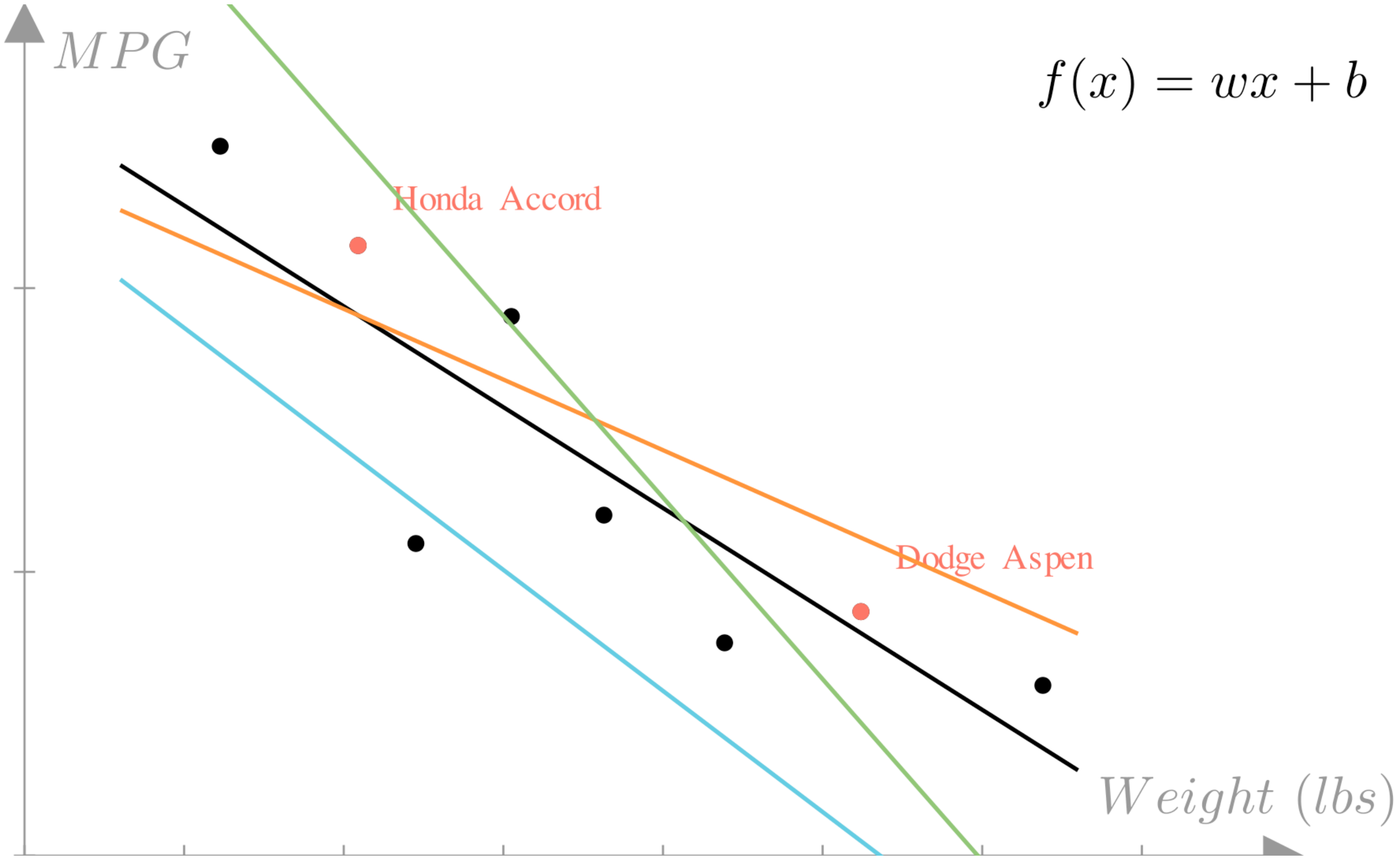
$$(\text{weight})w_1 + (\text{horsepower})w_2 + (\text{displacement})w_3 + (\text{0-60mph})w_4 + b$$

Or in matrix notation:

$$f(\mathbf{x}) = \underset{\substack{\uparrow \\ \text{Obs. car}}}{\begin{bmatrix} \text{Weight} \\ \text{Horsepower} \\ \text{Displacement} \\ \text{0-60mph} \\ 1 \end{bmatrix}} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ b \end{bmatrix}$$



$$f(x) = wx + b$$



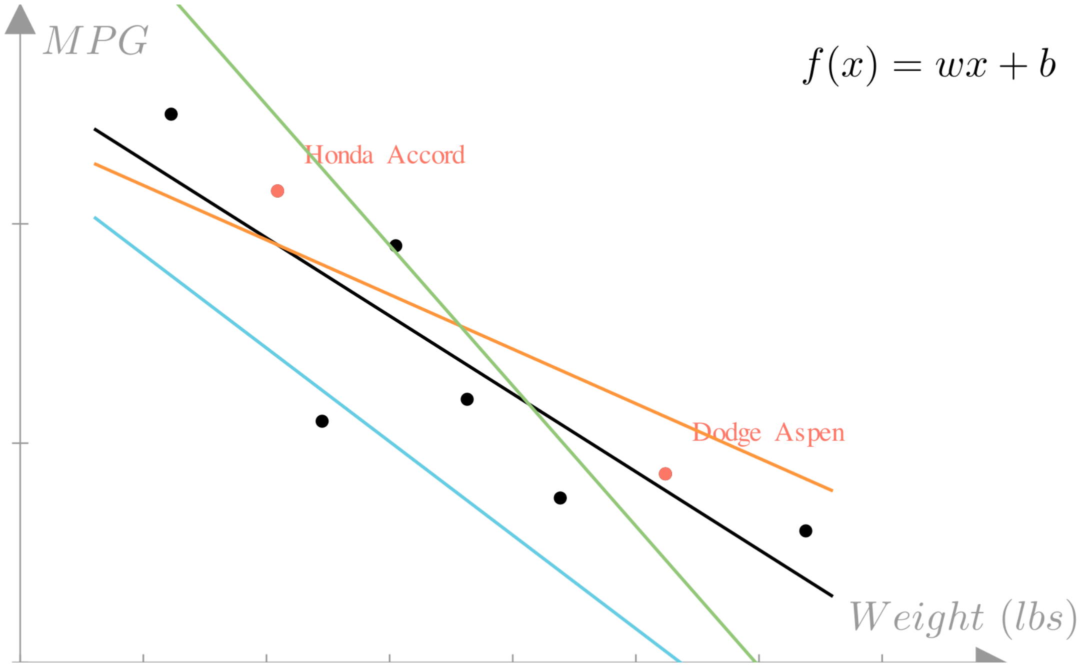
$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \end{bmatrix}$$

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i w_i + b$$

We typically refer to $\mathbf{w}$ specifically as the **weight vector** (or weights) and b as the **bias**. To summarize:

$$\textbf{Affine function: } f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b, \quad \textbf{Parameters:} \quad \big(\text{Weights: } \mathbf{w},\ \text{Bias: } b\big)$$

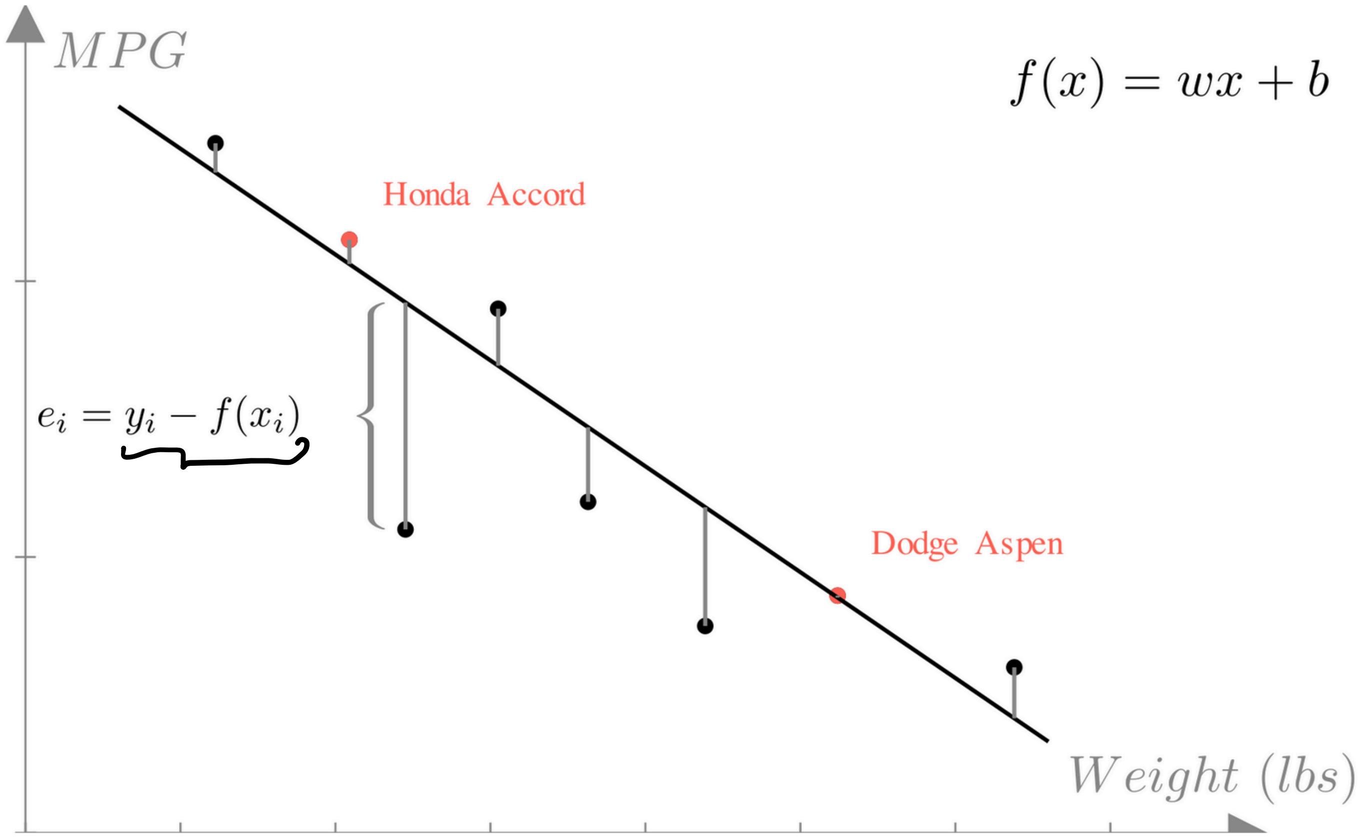
```
class Regression:
    def __init__(self, weights):
        self.weights = weights

    def predict(self, x):
        return np.dot(x, self.weights)
```



$$f(x) = wx + b$$

The **residual** or **error** of a prediction is the difference between the prediction and the true output:

$$e_i = y_i - f(\mathbf{x}_i)$$



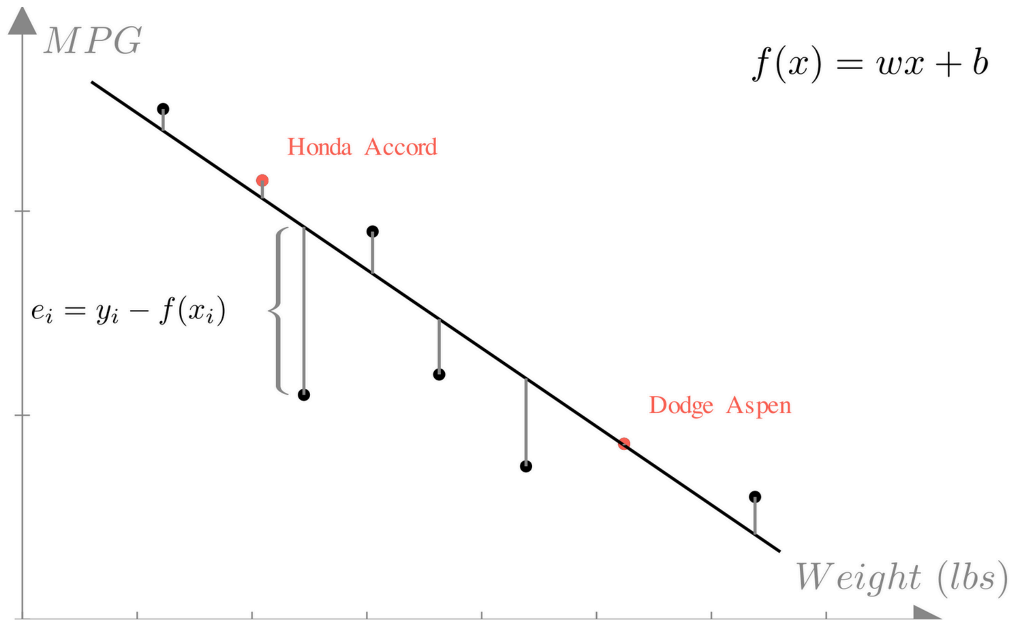
$$\mathcal{D} = \{(\mathbf{x}_1, y_1),\ (\mathbf{x}_2, y_2),\ \dots\ (\mathbf{x}_N, y_N)\}$$

Mean Squared error

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\underbrace{(\underbrace{f(\mathbf{x}_i)} - \underbrace{y_i})^2}_{(2} = \frac{1}{N}\sum_{i=1}^{N}(\underbrace{\mathbf{x}_i^T\mathbf{w}} - y_i)^2$$

N: # of dps

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f(x) - y_i|$$



$$\mathcal{D} = \{(\mathbf{x}_1, y_1),\ (\mathbf{x}_2, y_2),\ \dots\ (\mathbf{x}_N, y_N)\}$$

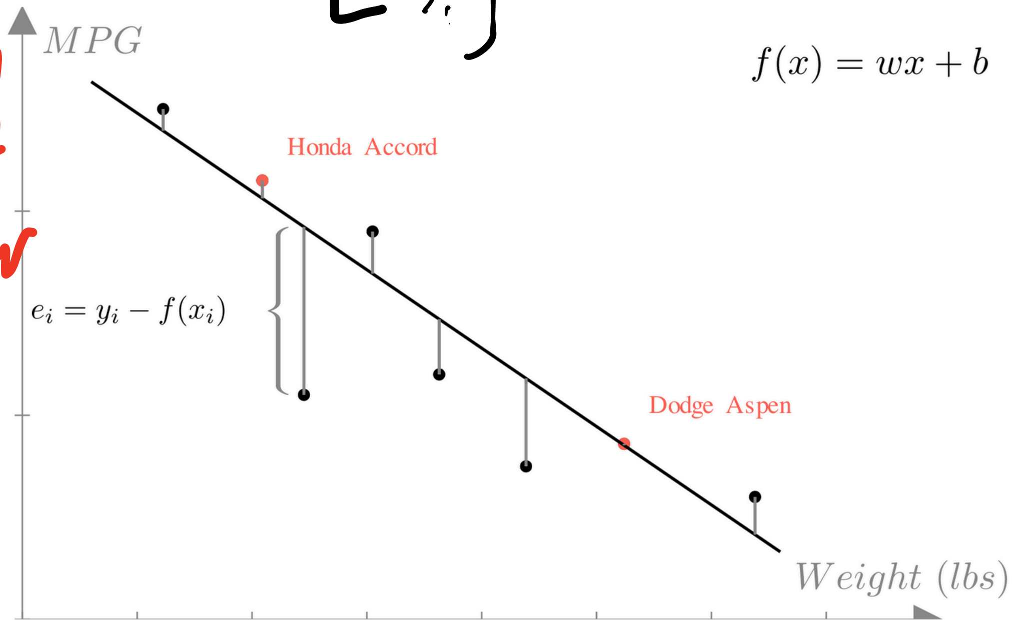
Vector

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_n \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{array}{cccc|l} \text{Weight} & \text{HP} & \cdots & \text{Displacement} & \\ x_{11} & x_{12} & \cdots & x_{1n} & \text{Obs. 1} \\ x_{21} & x_{22} & \cdots & x_{2n} & \text{Obs 2} \\ \vdots & \vdots & \ddots & \vdots & \\ x_{N1} & x_{N2} & \cdots & x_{Nn} & \text{Obs N} \end{array}, \quad \mathbf{y} = \begin{array}{c|l} y_1 & \text{Obs. 1} \\ y_2 & \text{Obs. 2} \\ \vdots & \\ y_N & \text{Obs. N} \end{array}$$

Capital bold X => Matrix of all Obs.

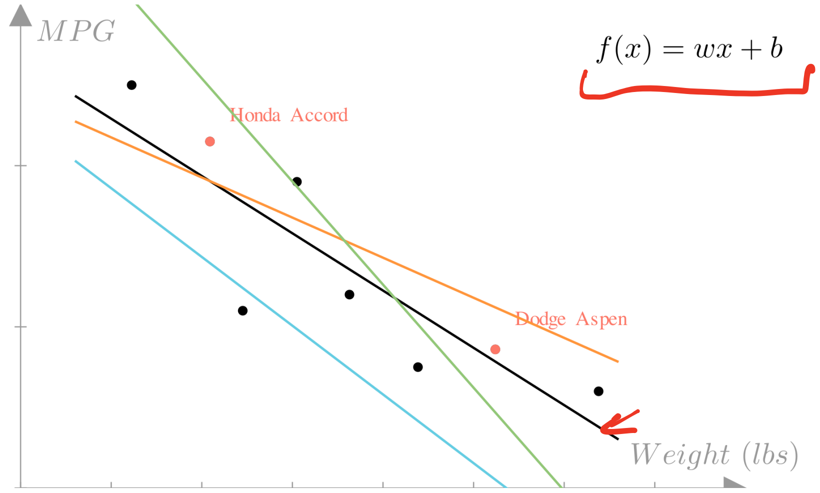
$$\underbrace{MSE}_{}(\underbrace{\mathbf{w}}_{\text{Parameters}}, \underbrace{\mathbf{X}}_{\text{Obs. Inputs}}, \underbrace{\mathbf{y}}_{\text{Obs. outputs/Labels}}) = \frac{1}{N}\sum_{i=1}^{N}(\underbrace{\mathbf{x}_i^T\mathbf{w}}_{\text{Pred.}} - \underbrace{y_i}_{\text{label}})^2$$

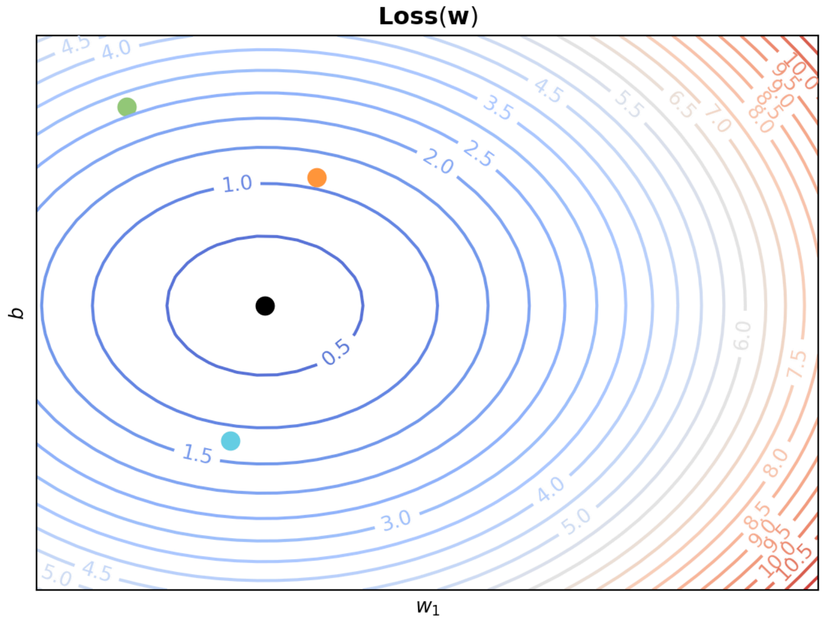


$$\underbrace{\mathbf{Loss}(\mathbf{w})} = MSE(\underset{\text{We can change } w}{\mathbf{w}}, \underbrace{\mathbf{X}, \mathbf{y}}_{\text{Fixed}}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i^T\mathbf{w} - y_i)^2$$

$$\mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}}\ \mathbf{Loss}(\mathbf{w}) = \underset{\mathbf{w}}{\text{argmin}}\frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i^T\mathbf{w} - y_i)^2$$

$$\underbrace{f(x) = wx + b}$$





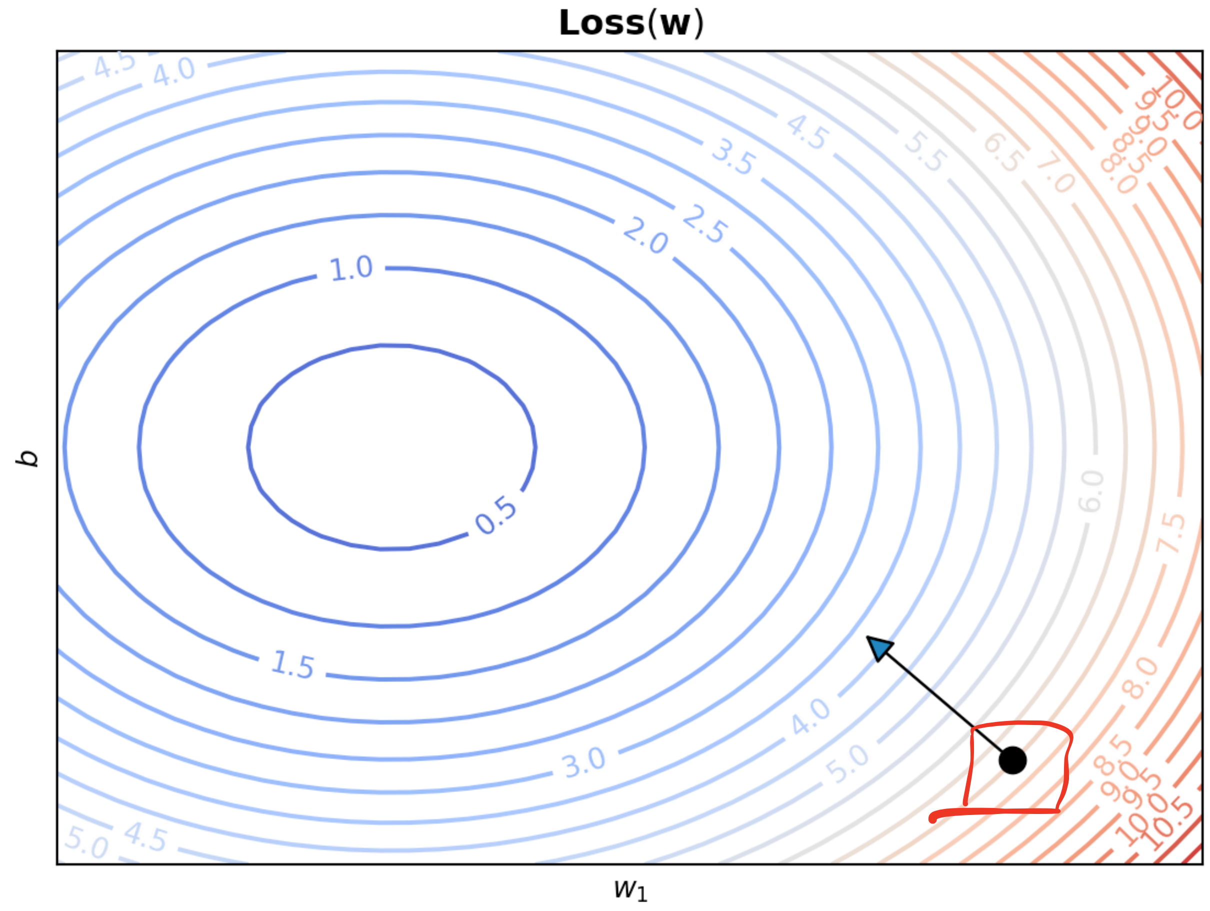
$$\text{Find: } \mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, f(\mathbf{w})$$

Improved guess

$$\mathbf{w}^{(1)} \leftarrow \boxed{\mathbf{w}^{(0)}} + \boxed{\mathbf{g}}$$

Initial guess

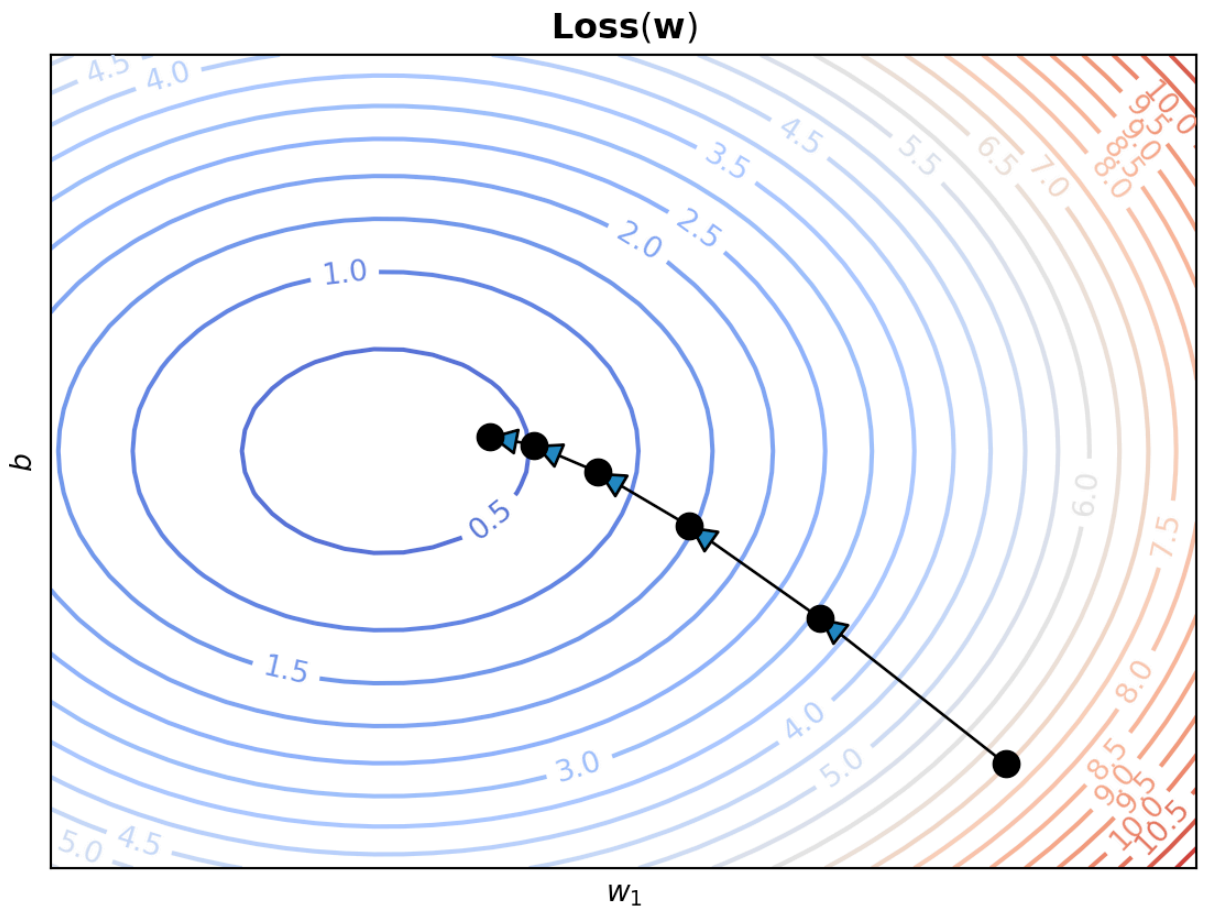
Direction of steepest decrease



$$\text{Find: } \mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, f(\mathbf{w})$$

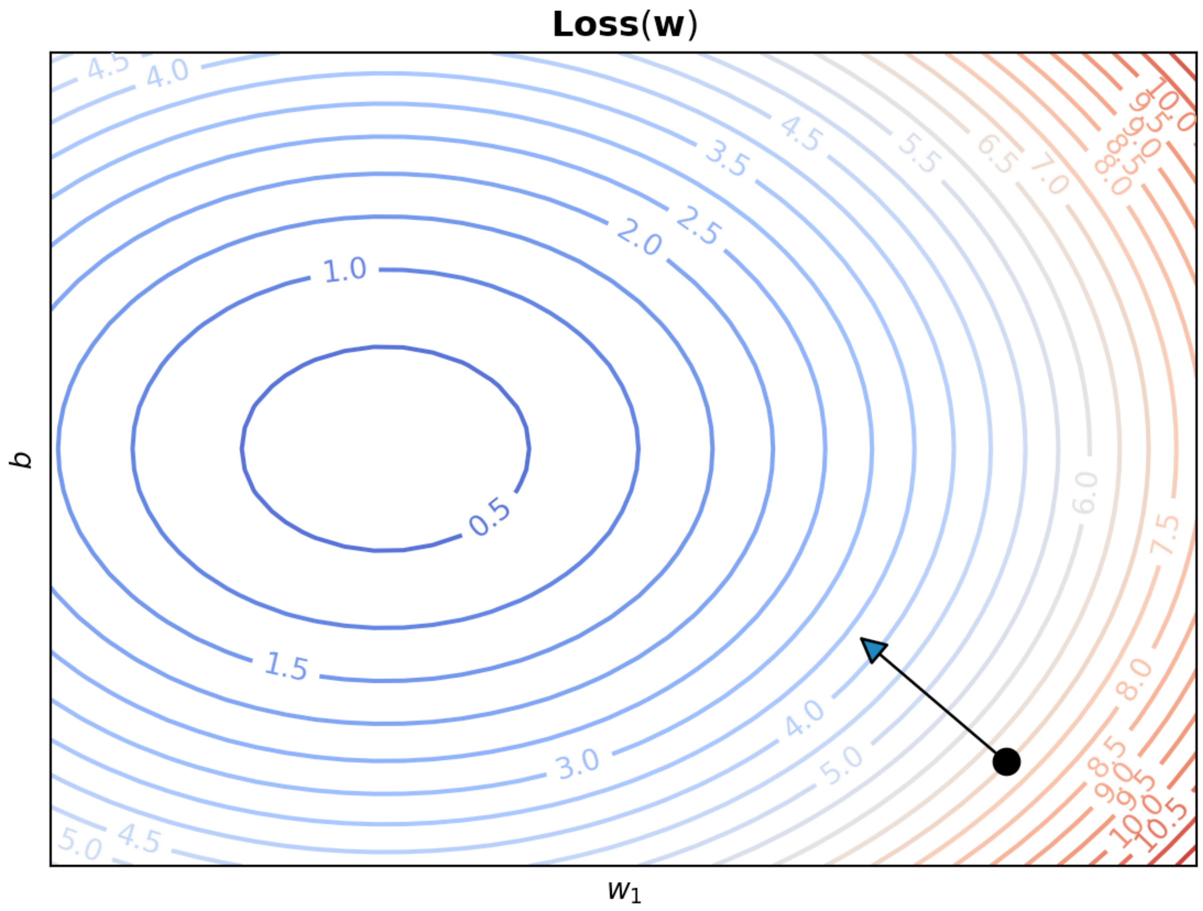
$$\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} + \mathbf{g}$$

gradient descent



$$\text{Find: } \mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \, f(\mathbf{w})$$

$$\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \nabla f(\mathbf{w}^{(0)})$$



The **gradient** of a vector-input function is a vector such that each element is the partial derivative of the function with respect to the corresponding element of the input vector. We'll use the same notation as derivatives for gradients.

$f(x)$

$$\frac{d f}{d \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \\ \vdots \end{bmatrix}$$

$f(x) = x^T x$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\frac{\partial}{\partial x_1} \mathbf{x}^T \mathbf{x} = \sum_{i=1}^{3} x_i \cdot x_i = \sum_{i=1}^{3} x_i^2 = x_1^2 + \cancel{x_2^2} + \cancel{x_3^2}$$

$$= 2x_1$$