Deep Learning

Part 2: Gradient Descent Method

Ali El Hadi Ismail Fawaz,
Germain Forestier, Hassan Ismail Fawaz

ENSISA 2A Info, Université Haute-Alsace

November 14, 2024

Table of Contents

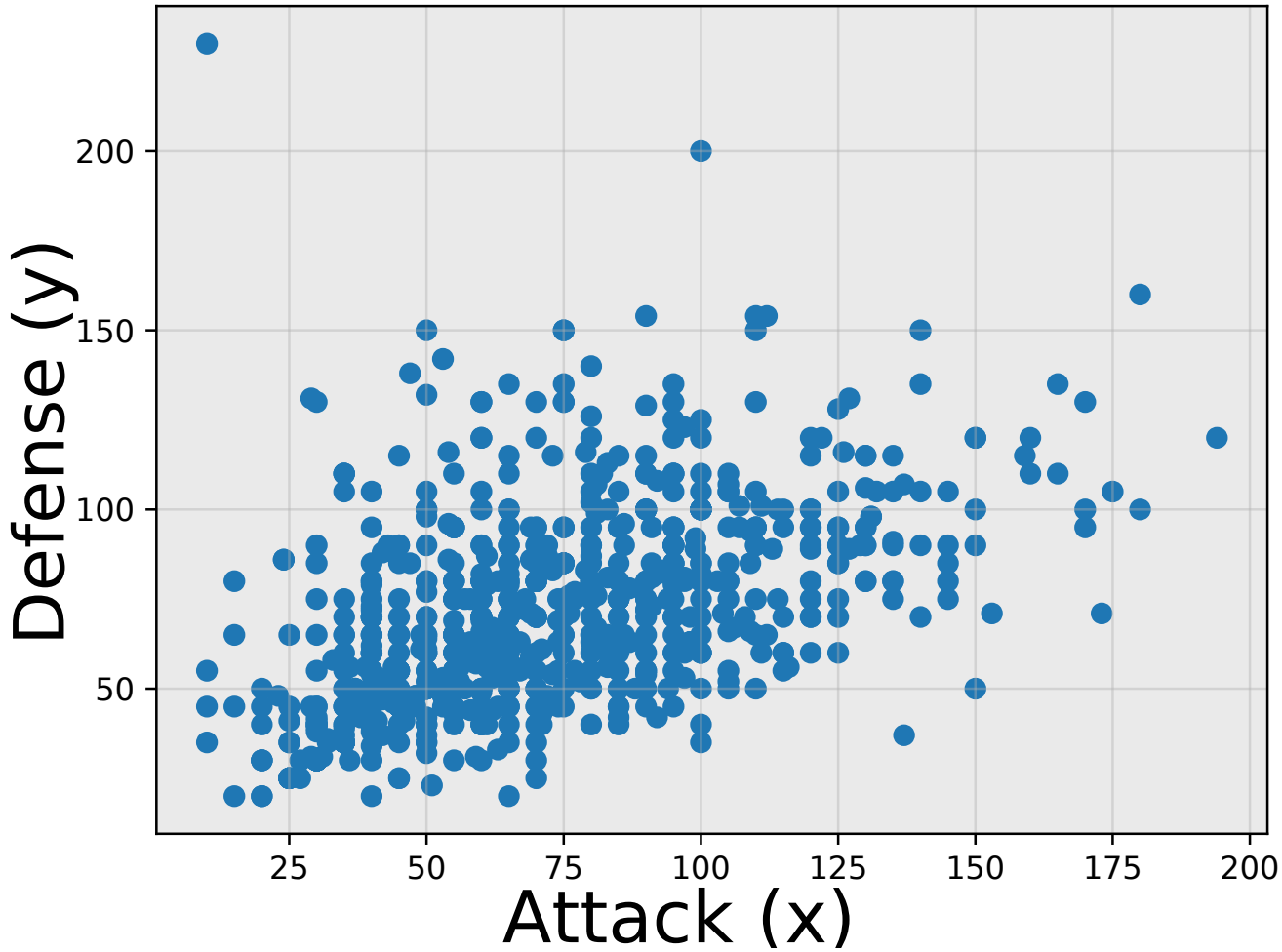
Preliminaries

Preliminaries

1. Define a model (e.g. linear function)
2. Define an objective (e.g. cost function)
3. Minimize the objective function:
   - Try with brute force (too long)
   - Optimal solution (derivative set to zero)
   - Gradient Descent (sign of derivative)

Preliminaries

Pokemon stats: Predict y in function of x



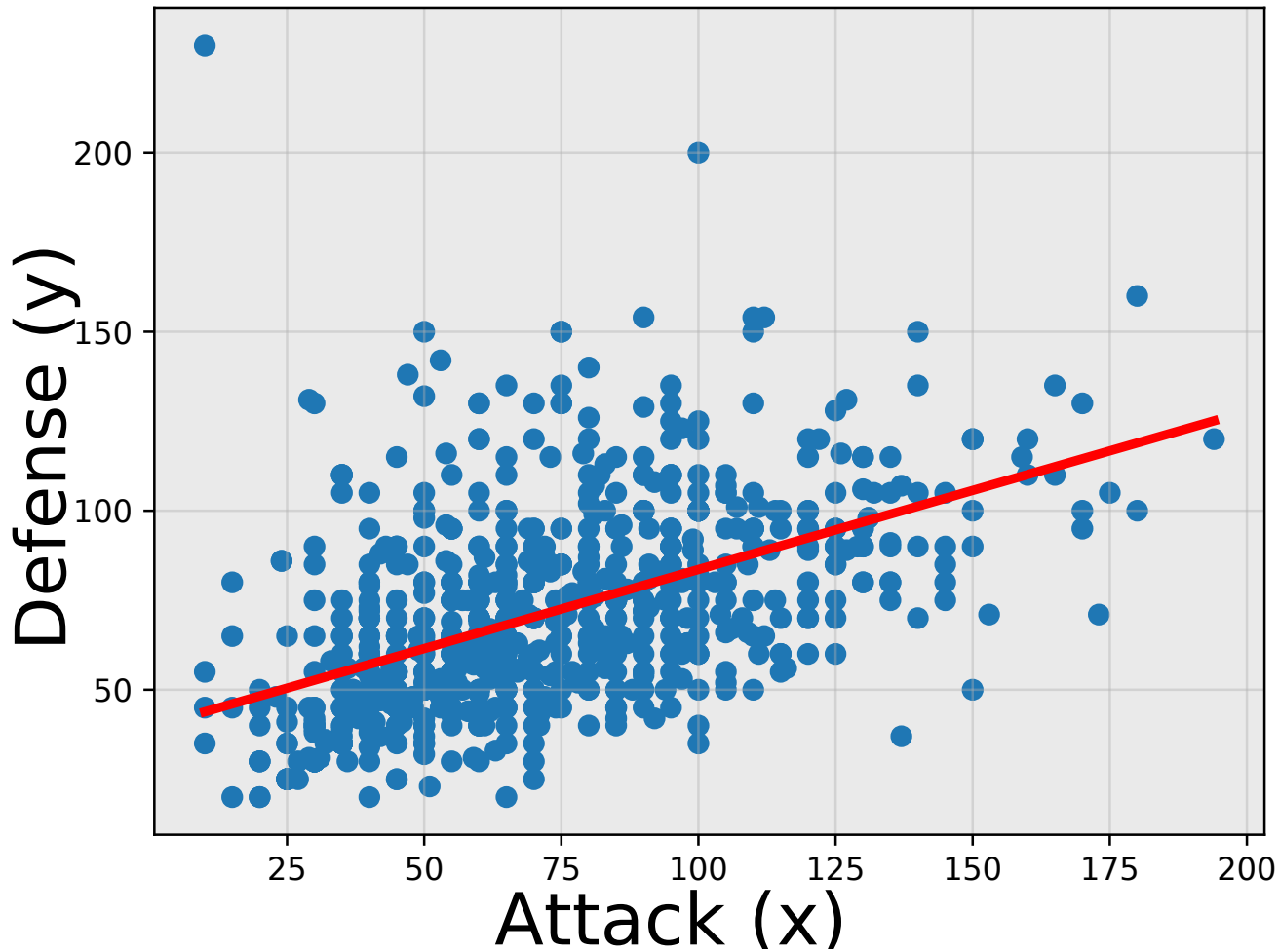
Preliminaries

A hypothesis:

$$\hat{y} = h(x) \tag{1}$$

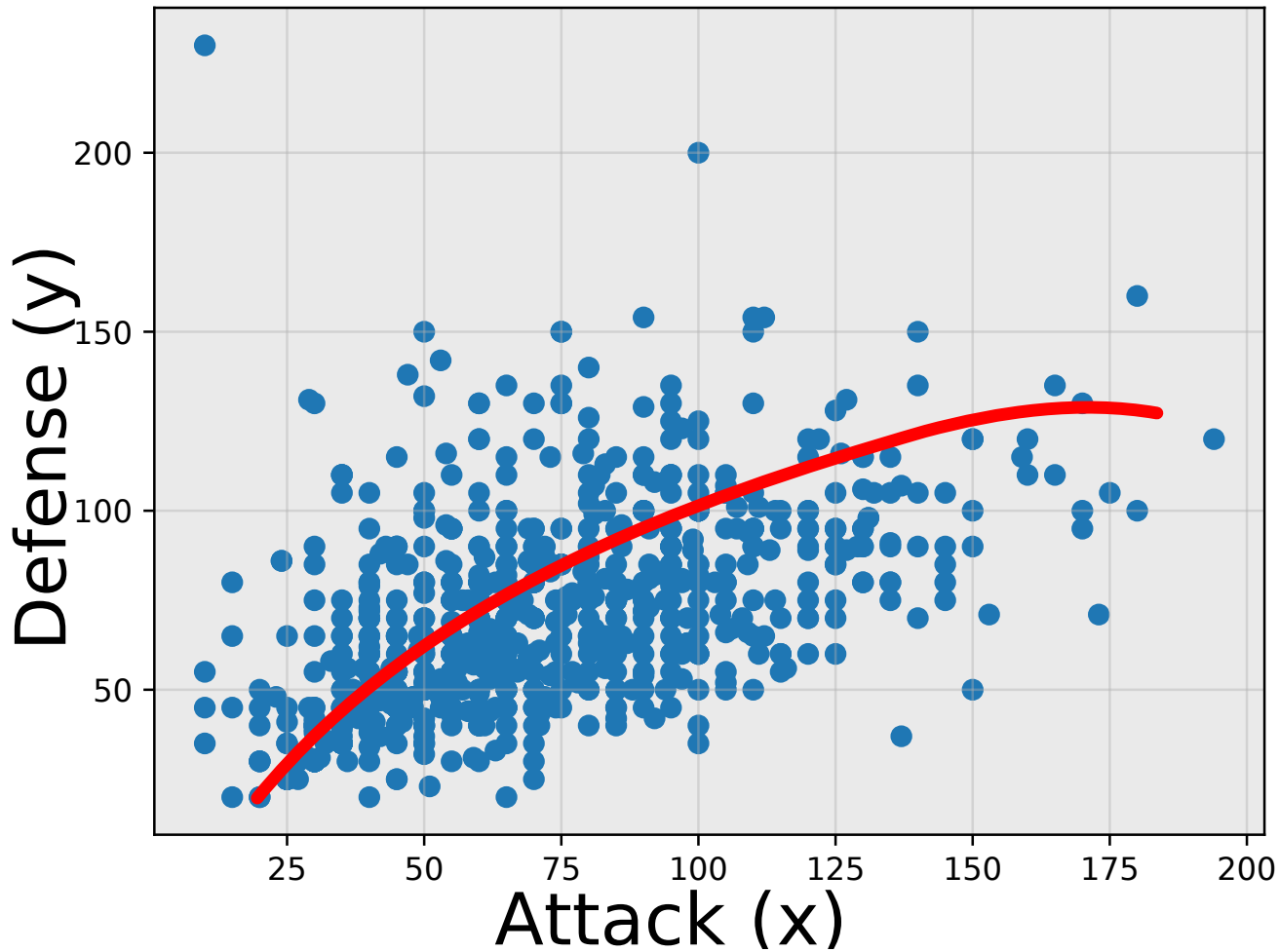
Preliminaries

Hypothesis: straight line $\hat{y} = h(x) = w.x + b$



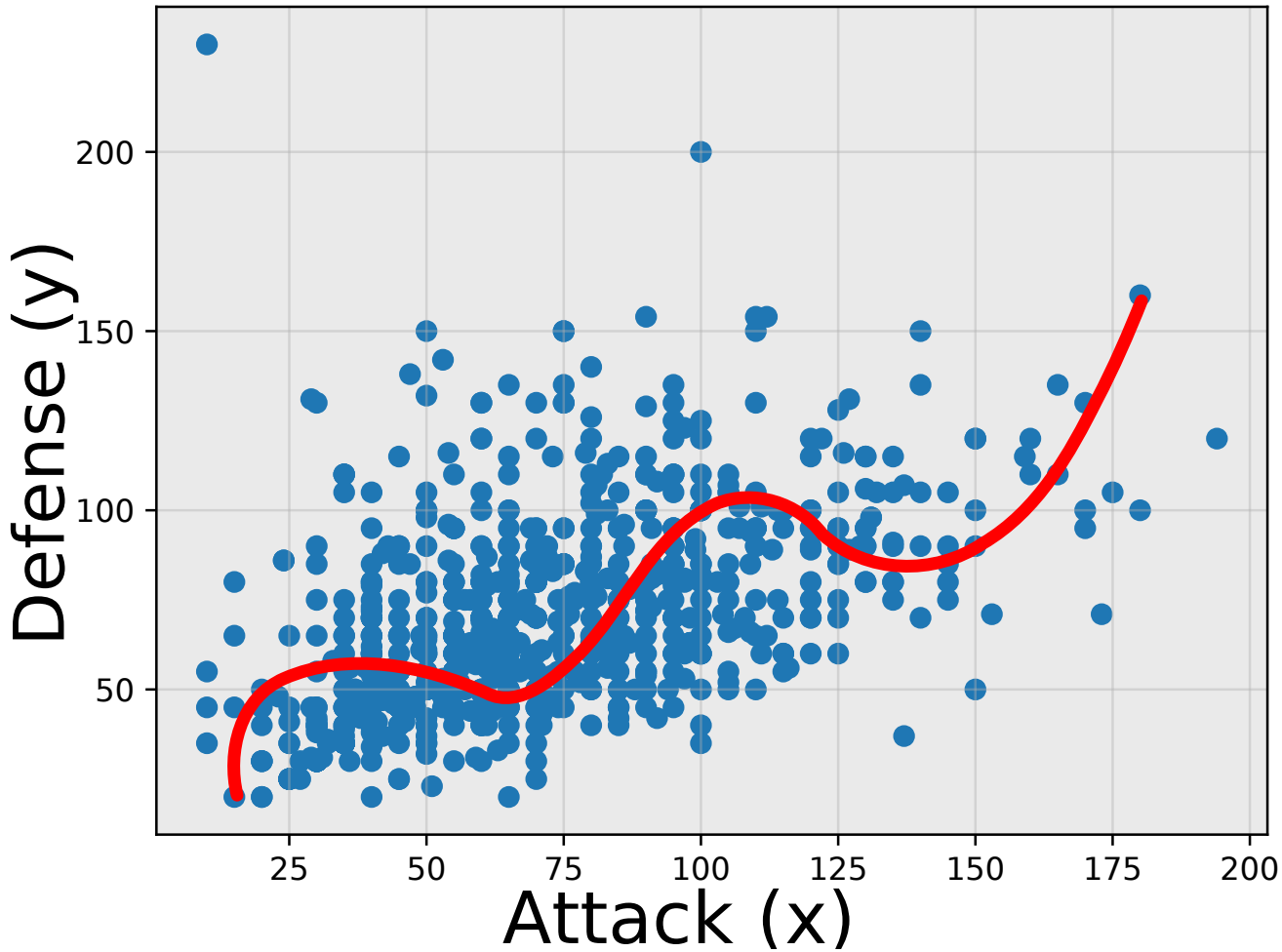
Preliminaries

Hypothesis: curve - 2_{nd} degree parabola $\hat{y} = h(x) = w_1.x^2 + w_2.x + b$



Preliminaries

Hypothesis: complex function $\hat{y} = h(x) = f(x, w_1, w_2, \ldots, w_n)$



Preliminaries

Define the objective

- Cost function:

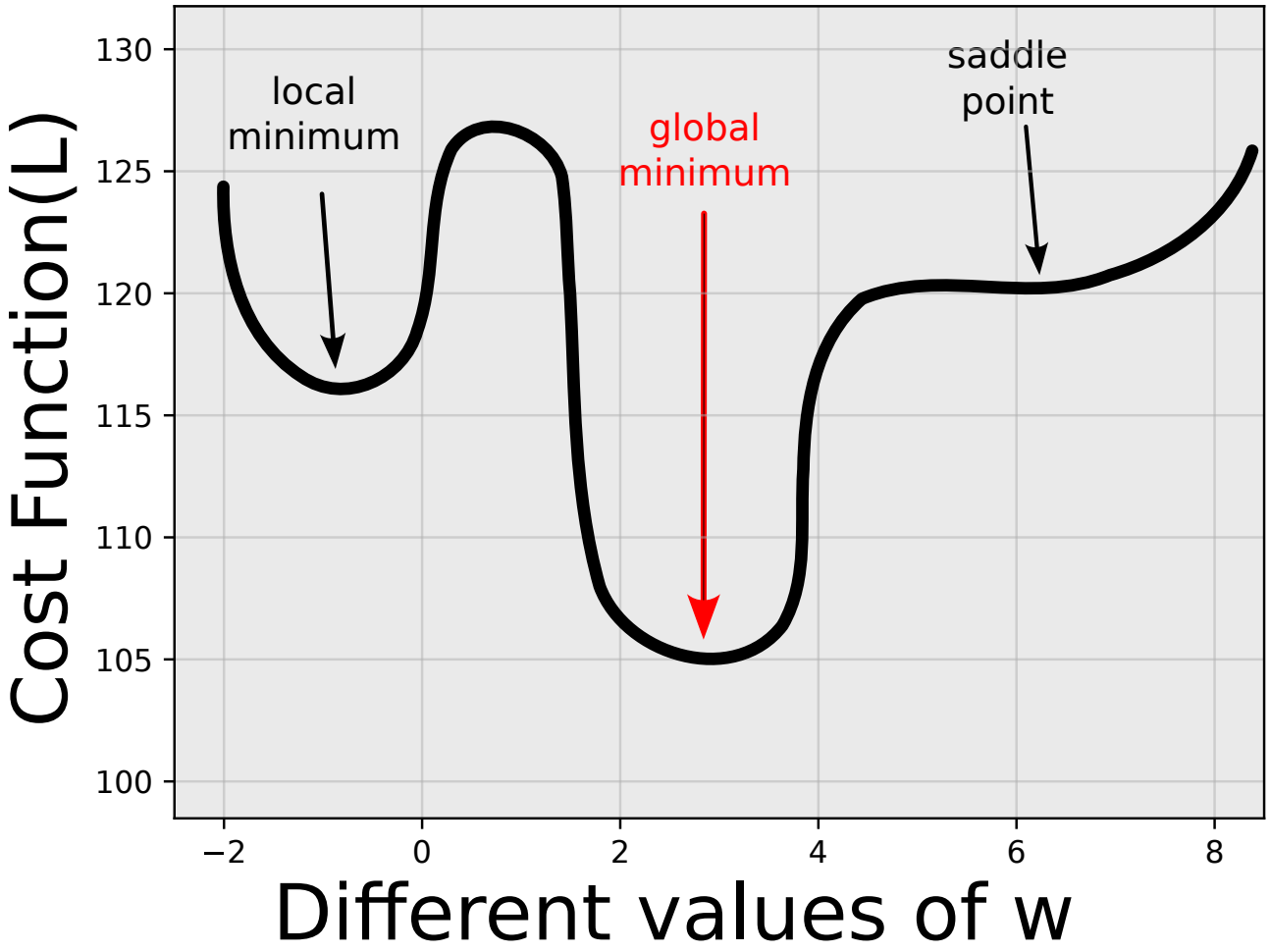
$$L(h) \tag{2}$$

- Objective:

$$\min_{h} L(h) \tag{3}$$

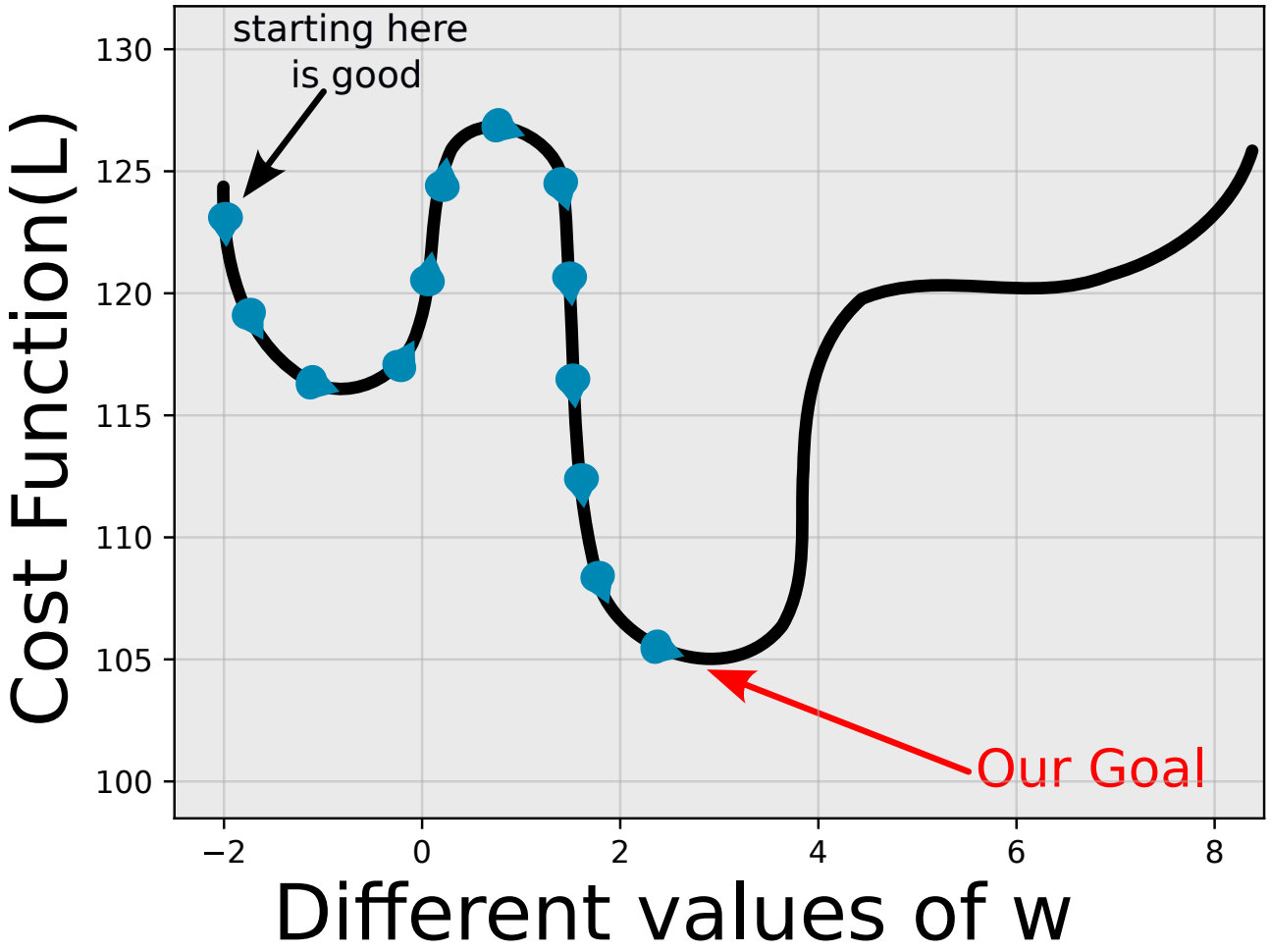
Preliminaries

Minimize a cost function



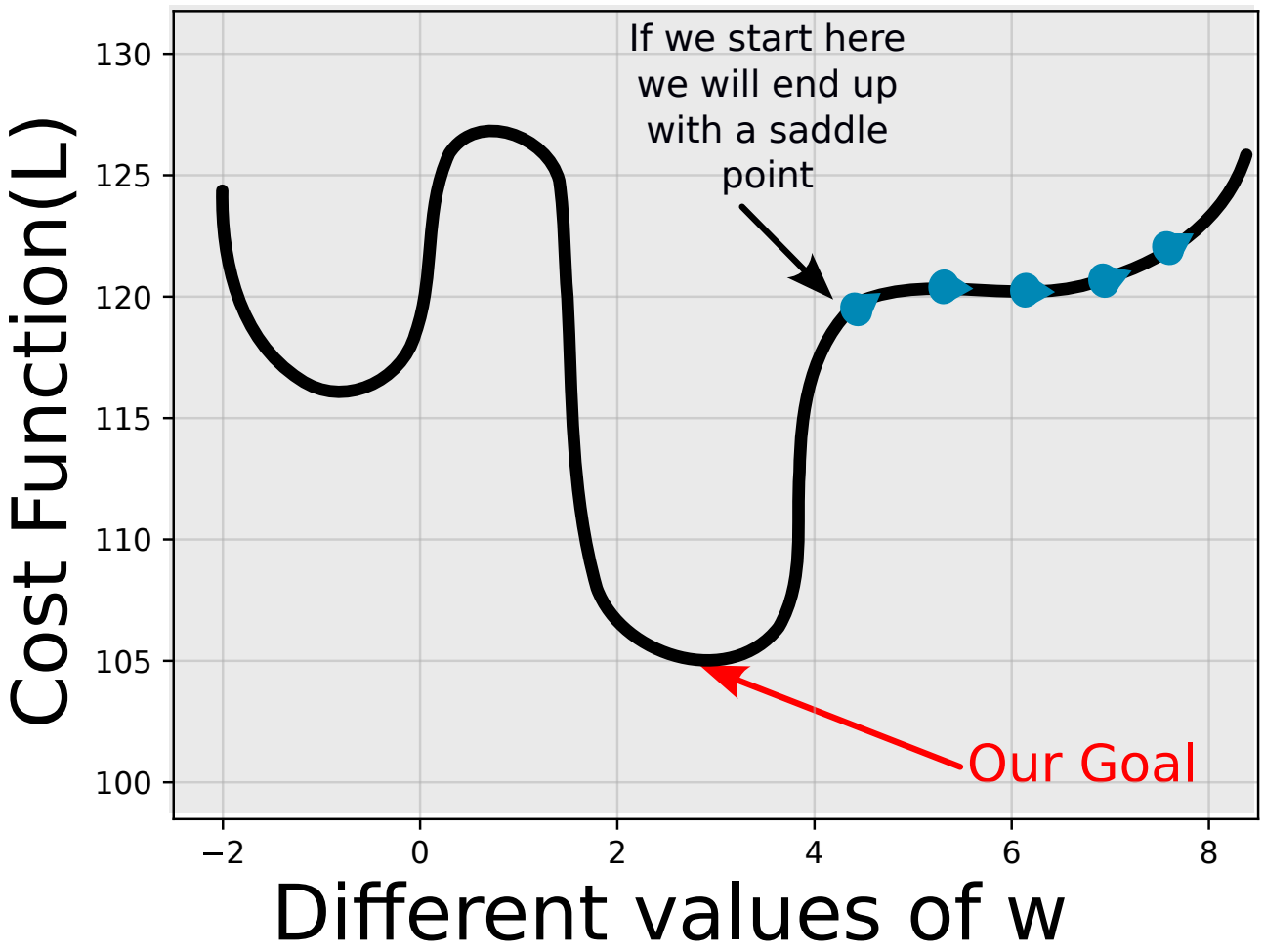
Preliminaries

Brute Force method



Preliminaries

Brute Force method



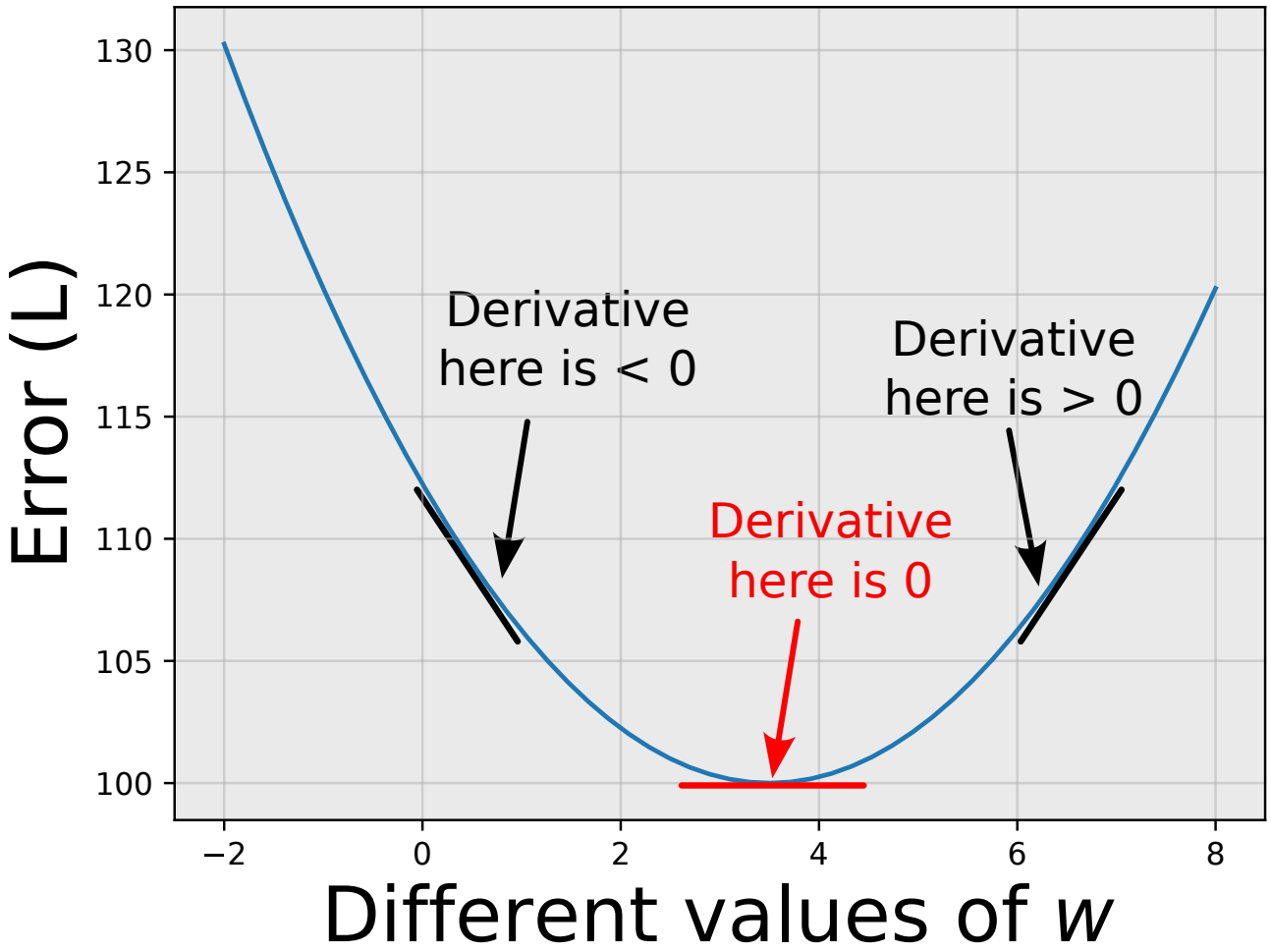
Preliminaries

Brute Force method is not to be used because:

- Very slow methid (if we have $w_1, w_2, \ldots, w_{1000}$
- Difficult to know the starting point
- We can end up with a bad minimum or even a saddle point

Preliminaries

Derivative (Gradient)



Preliminaries

Optimal Solution

- **Goal: Find w such that**

$$\frac{\partial L}{\partial w} = 0 \tag{4}$$

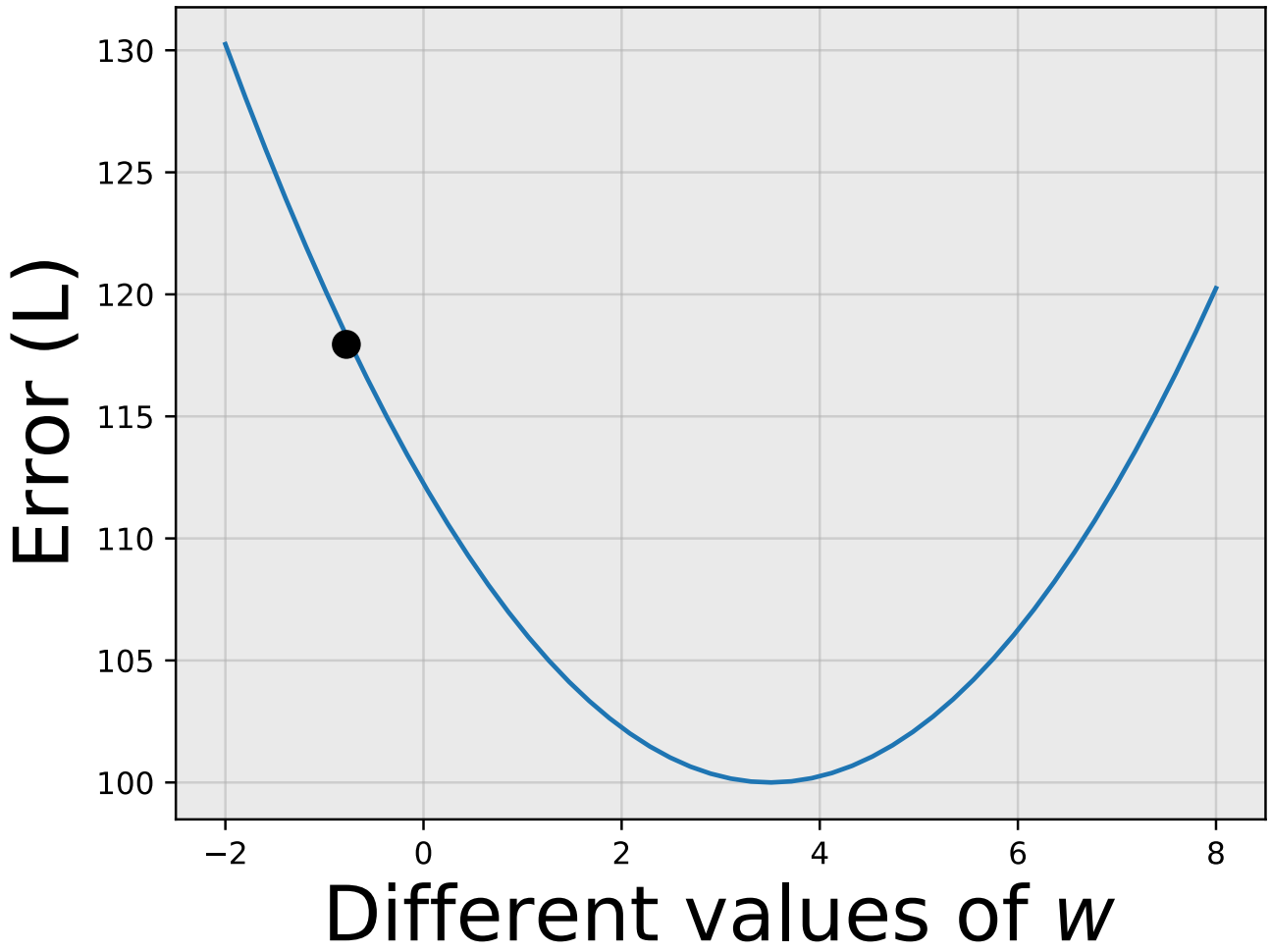
- Many methods exist to solve Equation 4:
  - Inverse normal equations
  - Orthogonal decomposition
- Limitations:
  - Very slow method especially in complex models
  - A problem if many equations exist for Equation 4

Gradient Descent

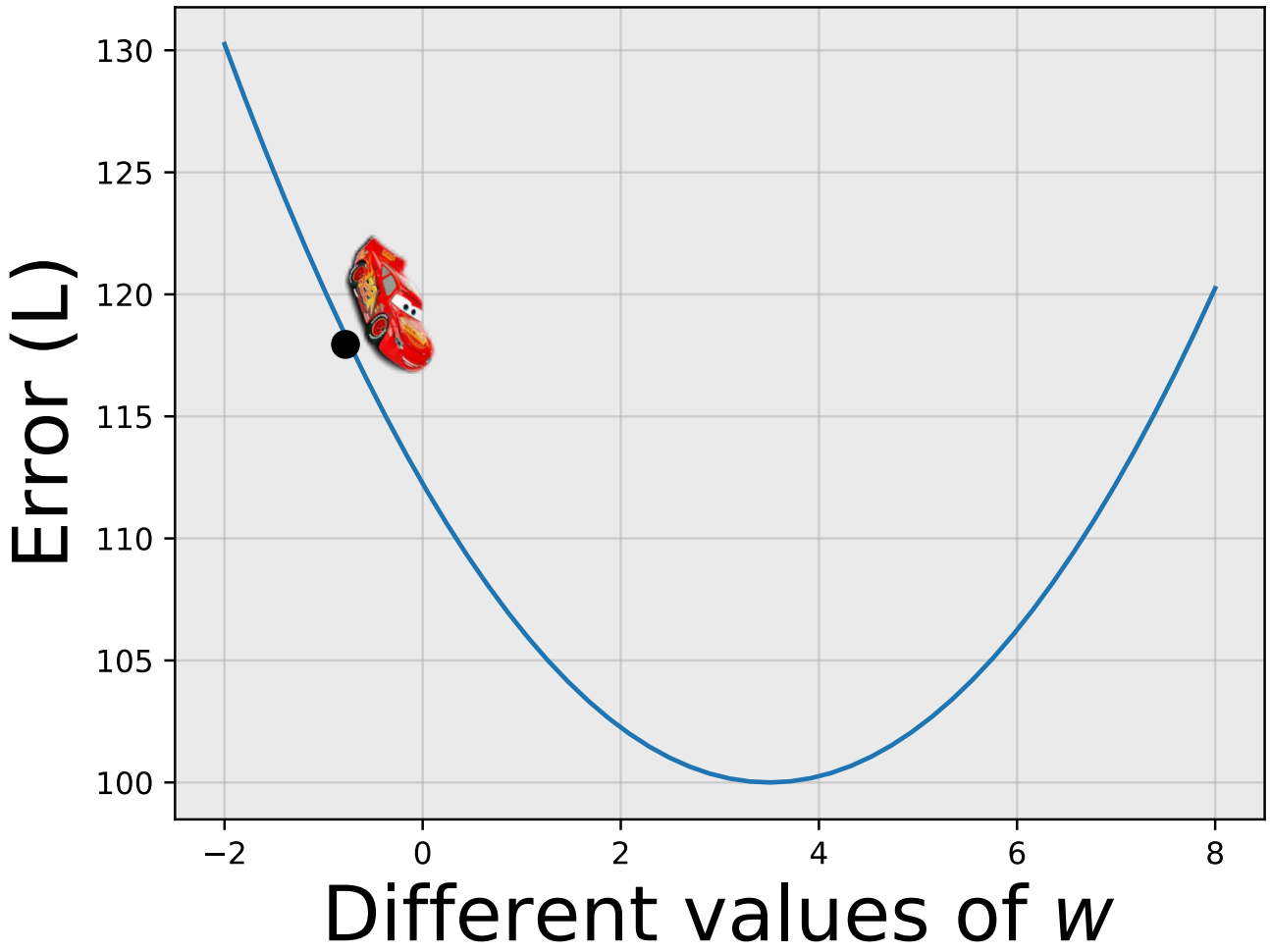
Gradient Descent

Our friend Ash is blindfolded and wants to arrive to the bottom of the valley to find Mewtwo

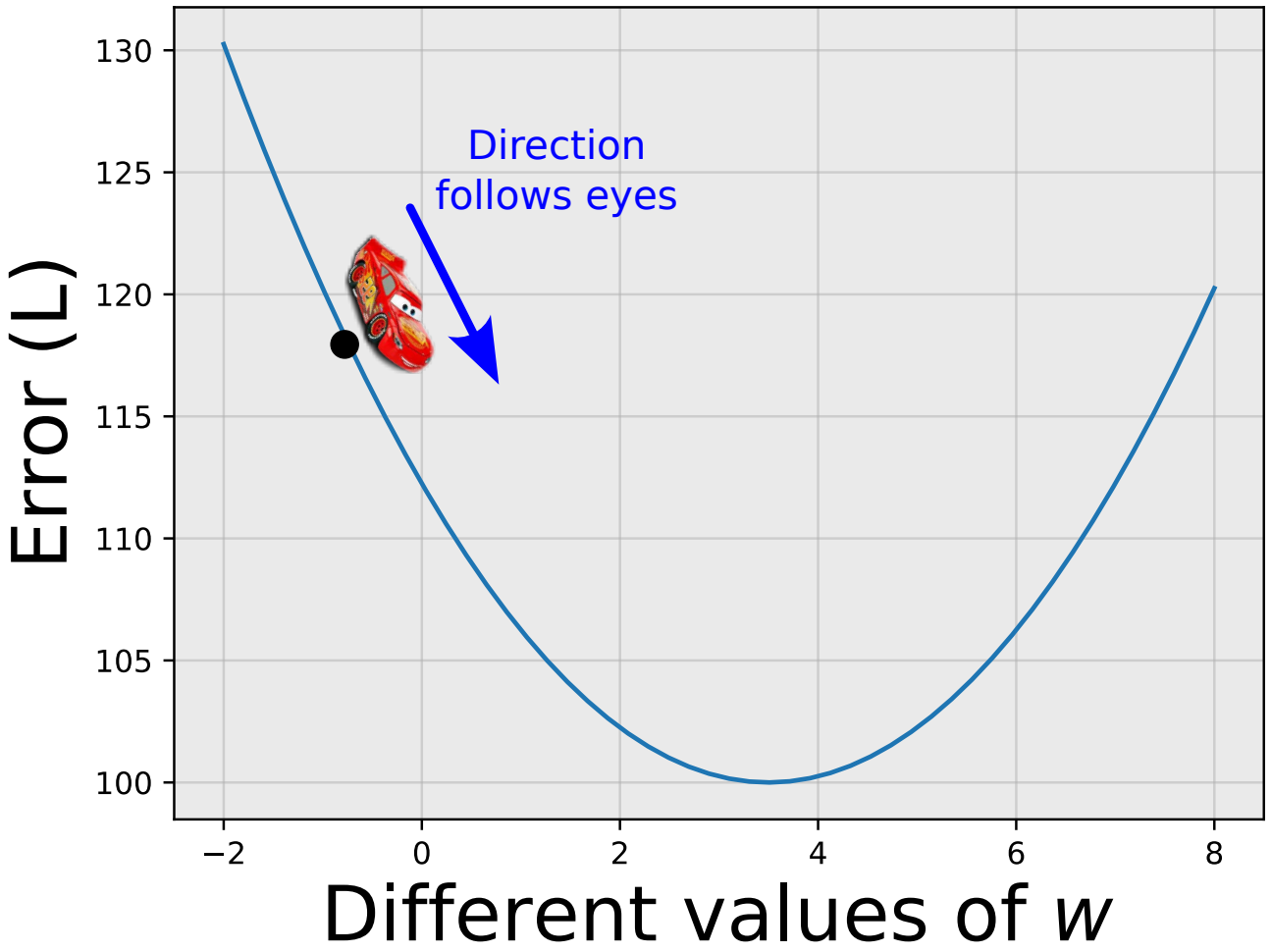
Gradient Descent



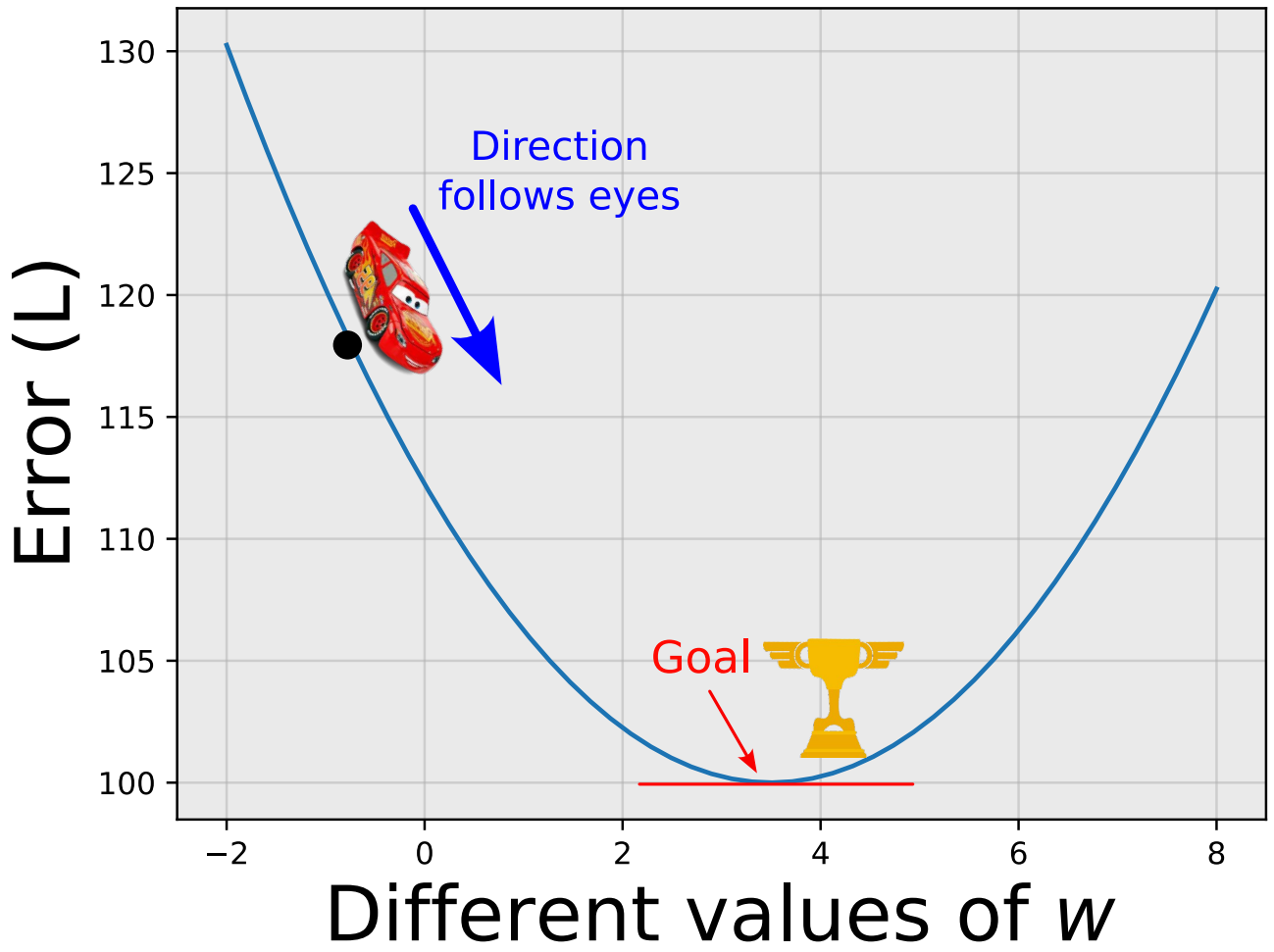
Gradient Descent



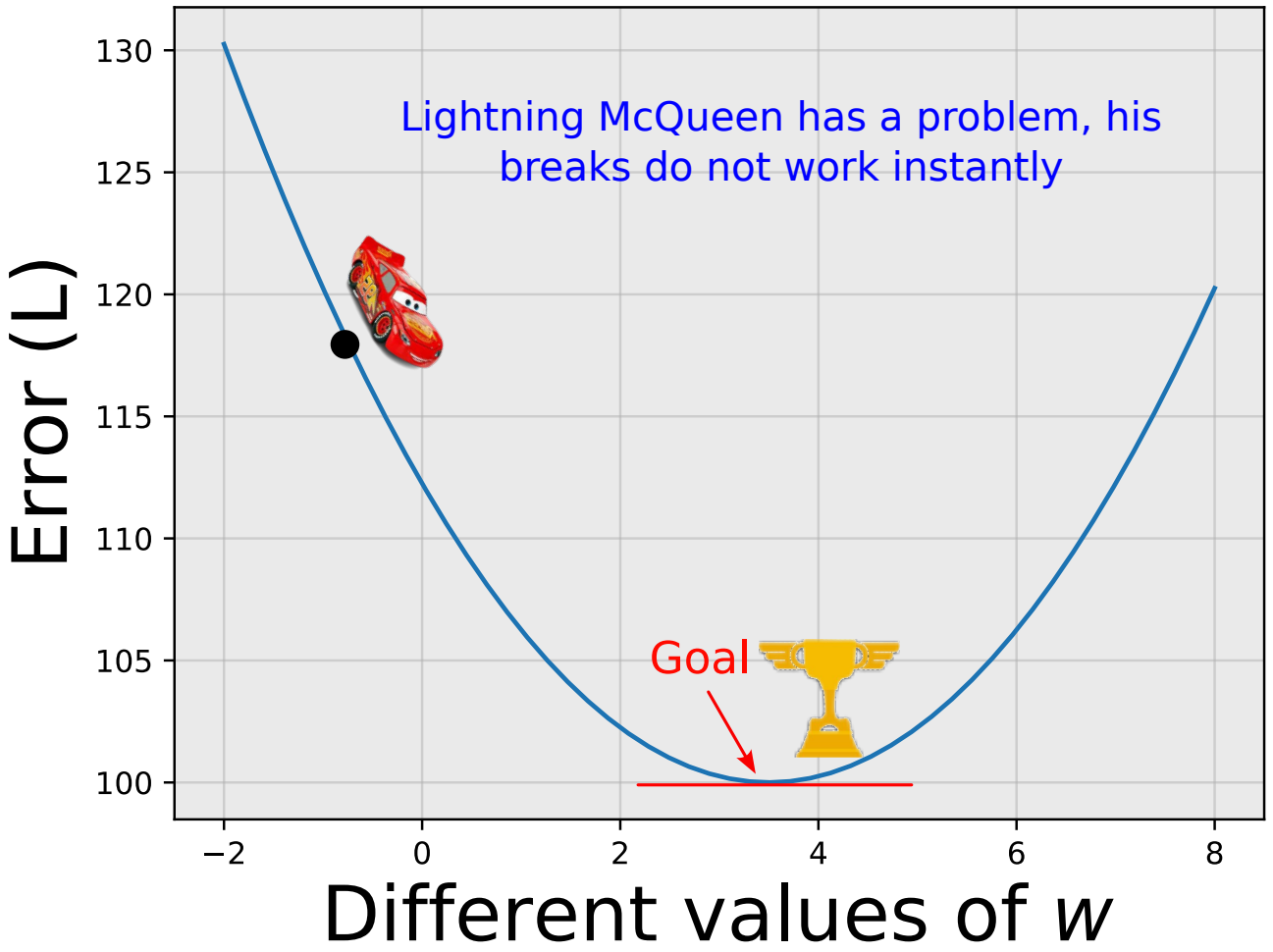
Gradient Descent



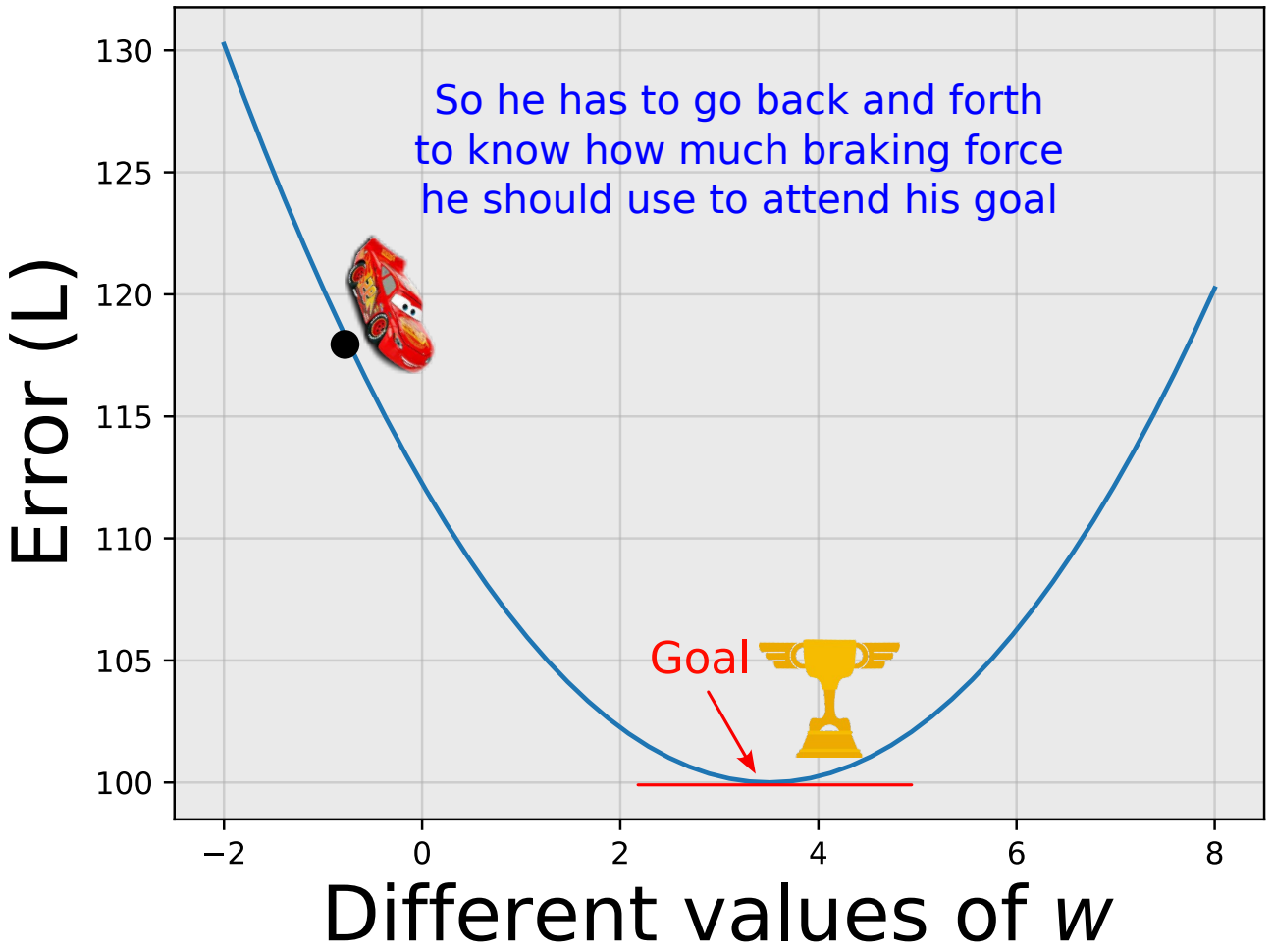
Gradient Descent



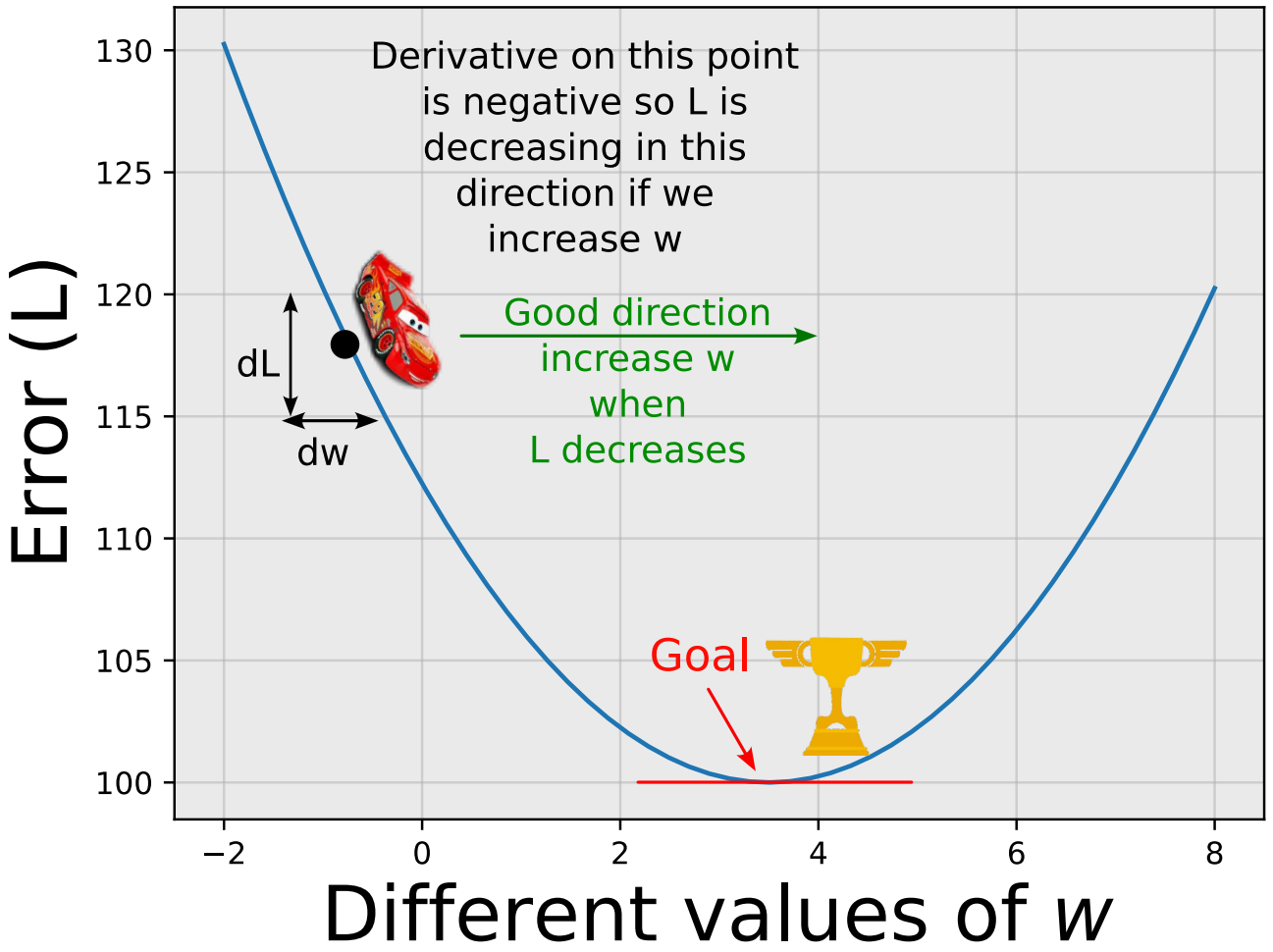
Gradient Descent



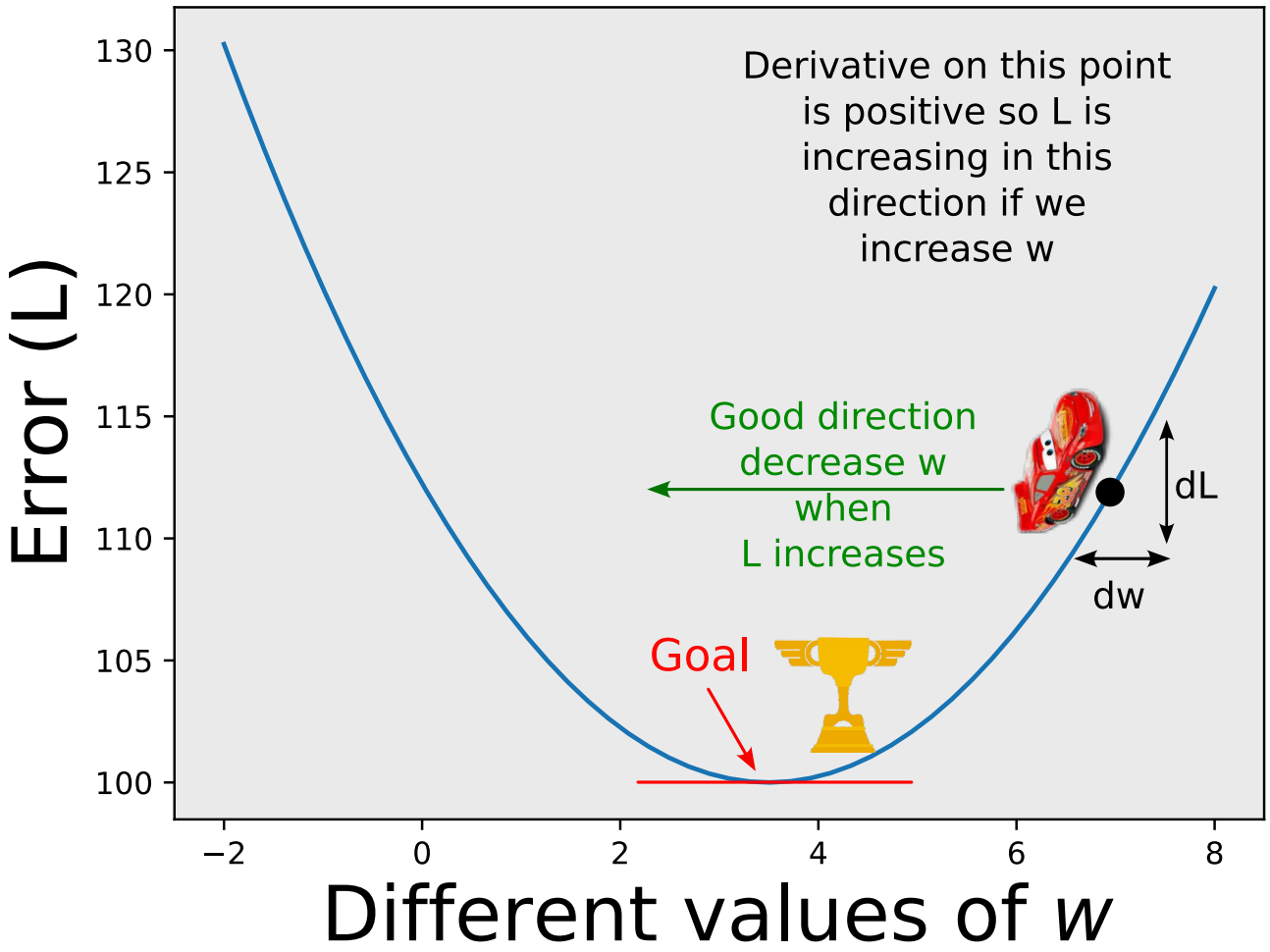
Gradient Descent



Gradient Descent



Gradient Descent



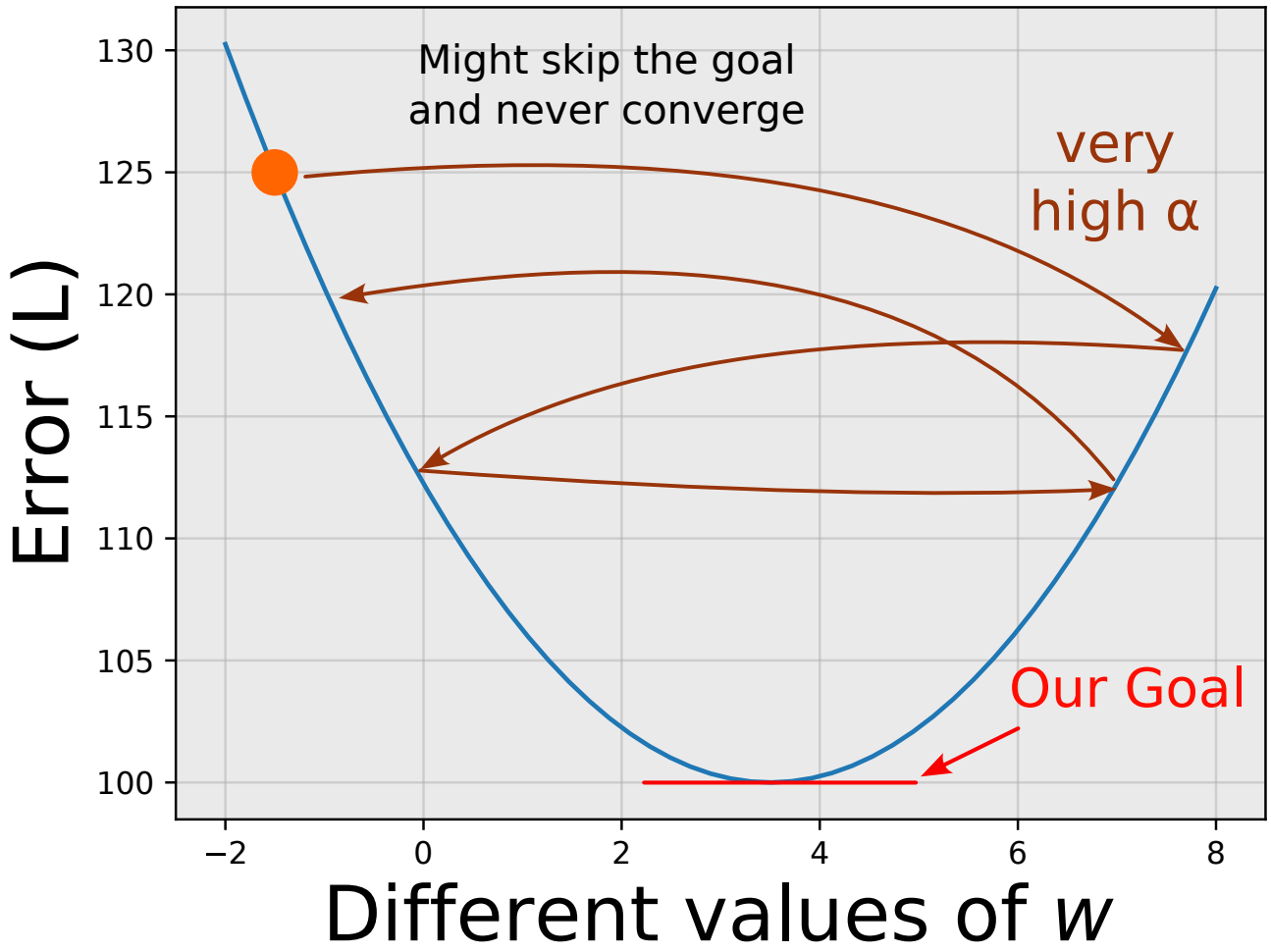
Gradient Descent

How to vaey w in a mathematical formula ?

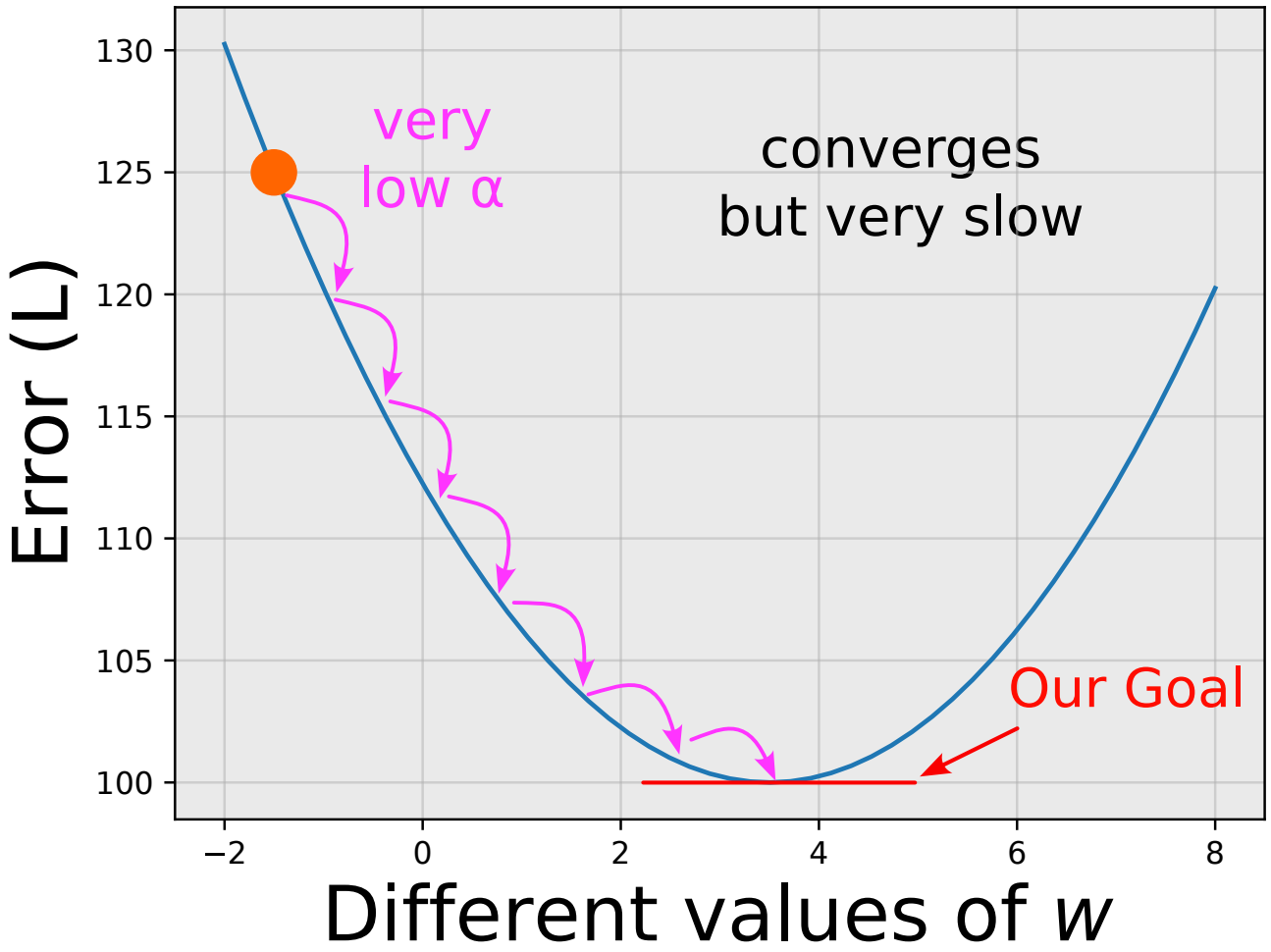
$$w = w - \alpha \frac{\partial L}{\partial w} \tag{5}$$

- w : the set of parameters to learn
- L : the cost function to minimize
- α : the learning rate that controls the variation of w
- $-$: The minus sign, means that we always vary w in the inverse sign of the gradient of L $\Longleftarrow$ minimize the error

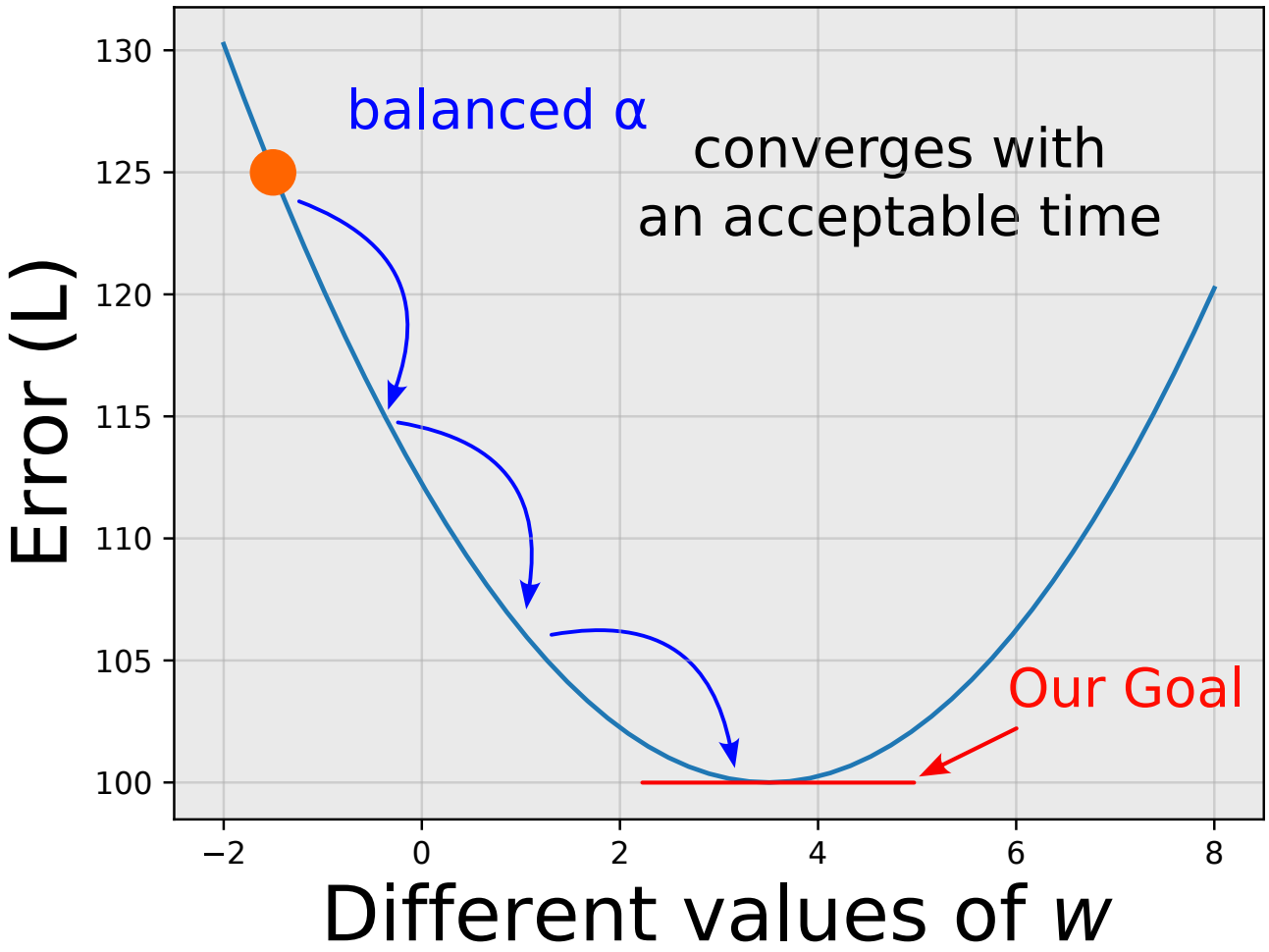
Gradient Descent



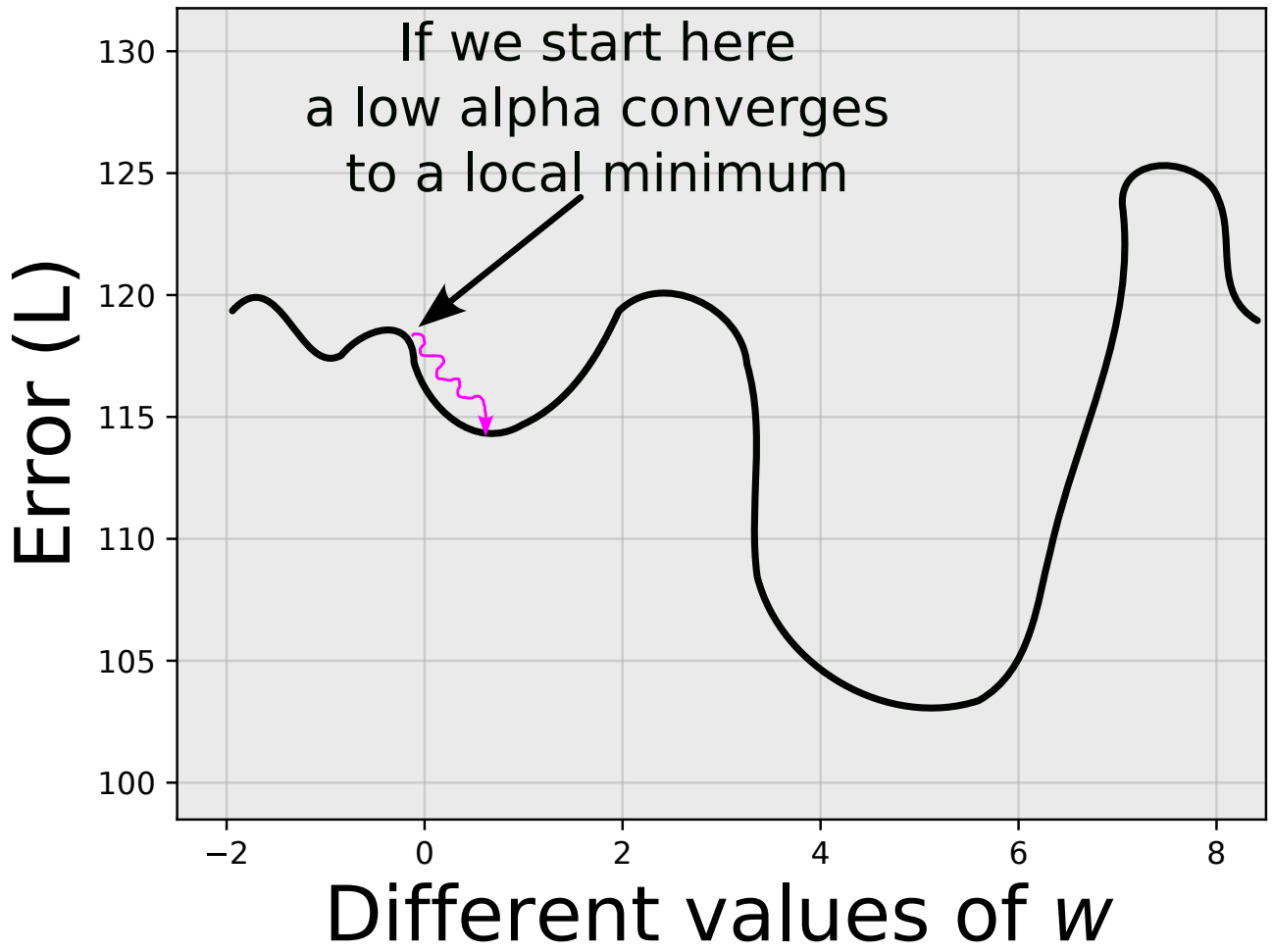
Gradient Descent



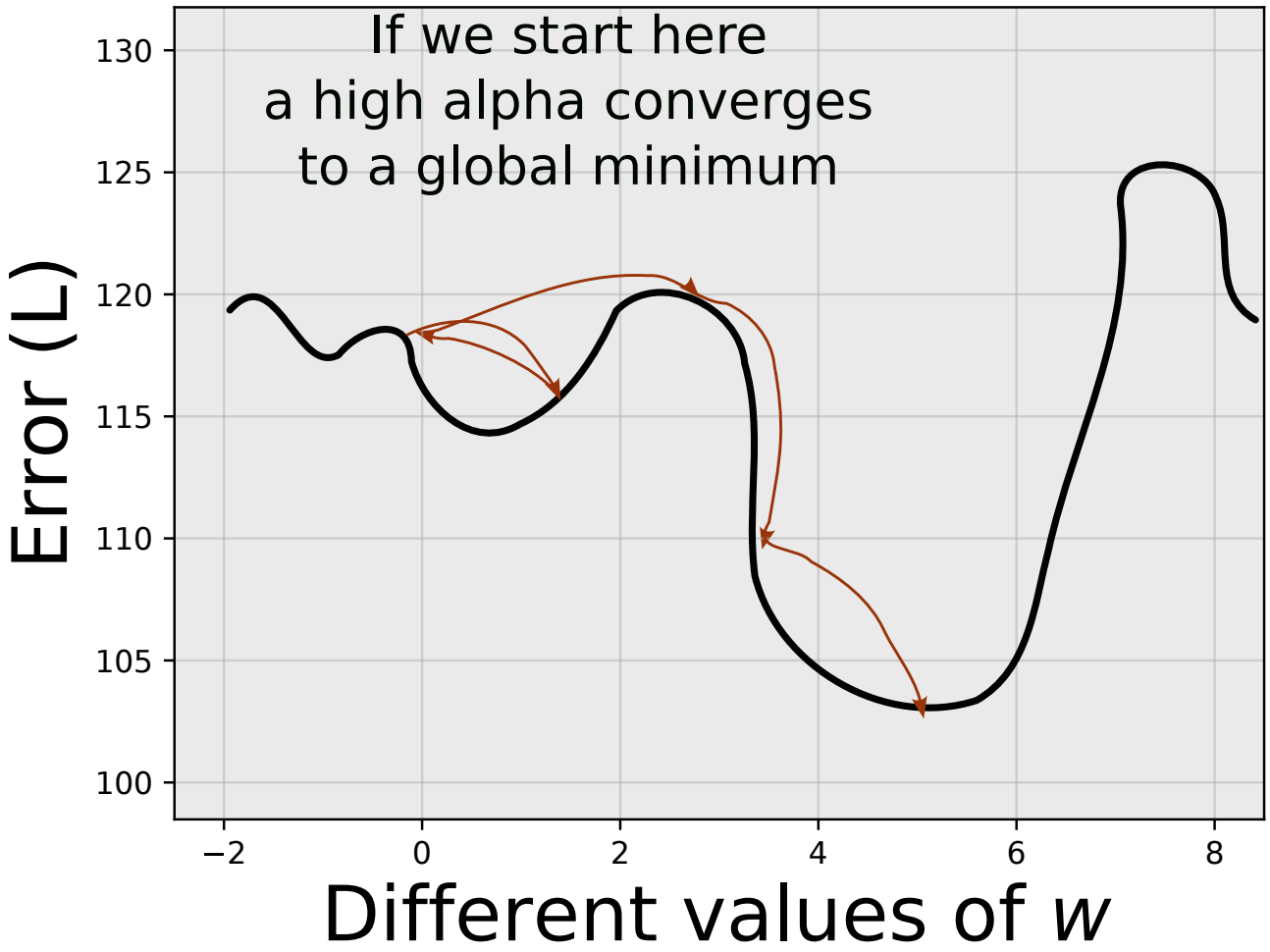
Gradient Descent



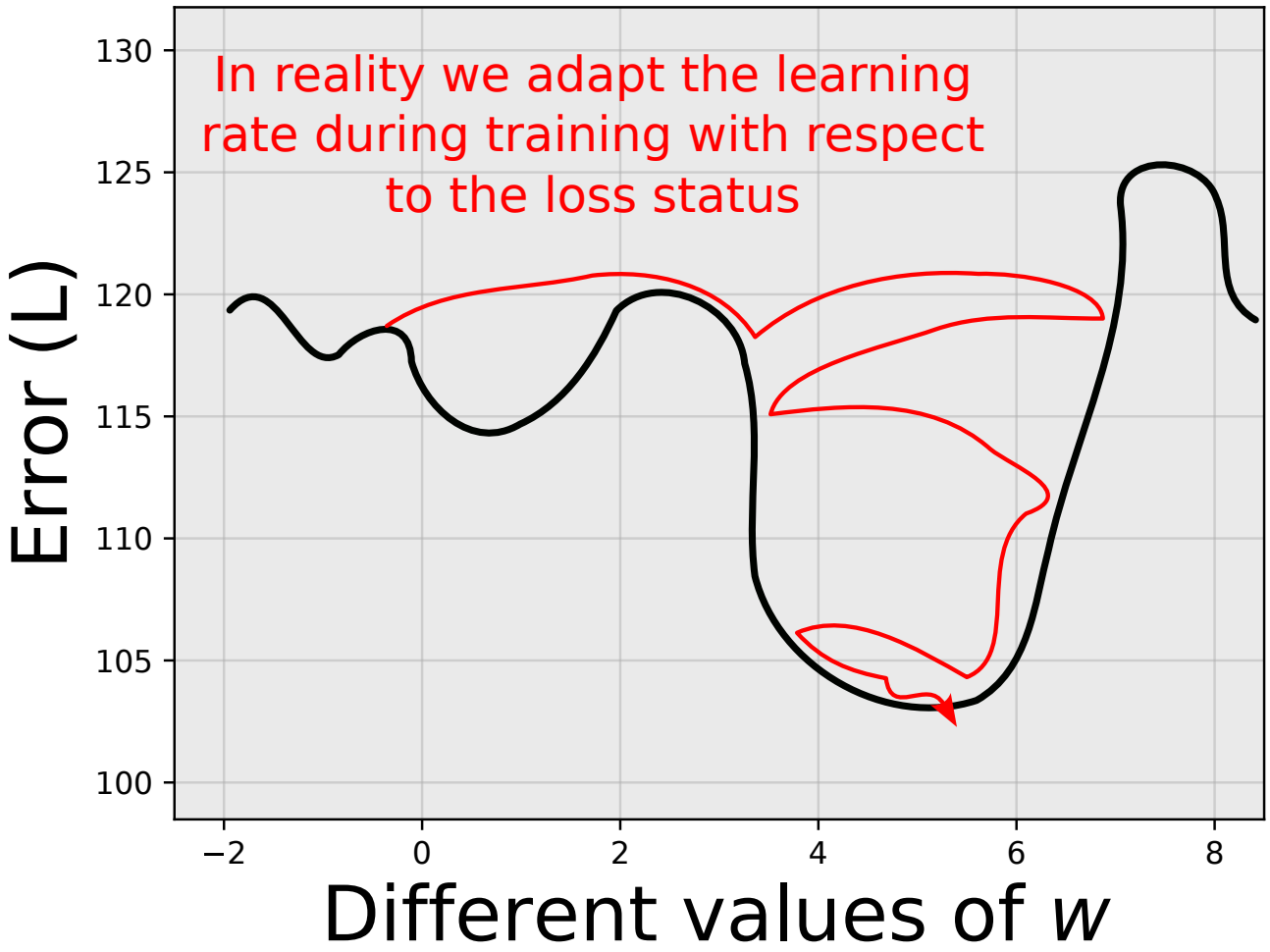
Gradient Descent



Gradient Descent



Gradient Descent



Gradient Descent

Mini-batch learning

- The algorithm "*Batch Gradient Descent*" optimizes the function:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} error(y_i, \hat{y}_i) \tag{6}$$

- The error should be calculated on all the training set, which is problematic if n is large

Gradient Descent

Mini-batch learning

- The algorithm "*Batch Gradient Descent*" optimizes the function:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} error(y_i, \hat{y}_i) \tag{6}$$

- The error should be calculated on all the training set, which is problematic if n is large

$\Longrightarrow$ "mini-batch" concept

$\Longrightarrow$ calculate L for $m <<< n$

$$L(w) = \frac{1}{m} \sum_{i=s}^{s+m-1} error(y_i, \hat{y}_i) \ \forall \ s \in \{1.m.2m, \ldots, n-m\} \tag{7}$$

Gradient Descent

Mini-batch learning

- The algorithm "*Batch Gradient Descent*" optimizes the function:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} error(y_i, \hat{y}_i) \tag{6}$$

- The error should be calculated on all the training set, which is problematic if n is large

$\Longrightarrow$ "mini-batch" concept

$\Longrightarrow$ calculate L for $m <<< n$

$$L(w) = \frac{1}{m} \sum_{i=s}^{s+m-1} error(y_i, \hat{y}_i) \ \forall \ s \in \{1.m.2m, \ldots, n-m\} \tag{7}$$

- Once we go through all batches = 1 epoch
- We repeat the procedure many times (many epochs)

Gradient Descent

Pseudo-code for the min-batch gradient descent

Input: X, Y the training dataset
Output: W the parameters that represents the relation $X \implies Y$

Random initialisation of the parameters W

for epoch $= 1 \dots 1000$ **do**

Randomly shuffle the dataset

for each mini-batch X_m, Y_m of size m **do**

$$predictions = model(X_m, W)$$
$$errors = error_function(Y_m, predictions)$$
$$gradient = \frac{\partial L(W)}{\partial W}$$
$$W = W - \alpha.gradient$$

end for

end for

return W

Gradient Descent

Let m be the size of a min-batch

- If $m = n \implies$ Batch Gradient Descent
  - We should calculate the error and gradient on all training examples to change W each epoch

Gradient Descent

Let m be the size of a min-batch

- If $m = n \implies$ Batch Gradient Descent
  - We should calculate the error and gradient on all training examples to change W each epoch
- If $m = 1 \implies$ Stochastic Gradient Descent
  - We lose the parallelization of the calculation on multiple examples (and we have more error)

Gradient Descent

Let m be the size of a min-batch

- If $m = n \implies$ Batch Gradient Descent
  - We should calculate the error and gradient on all training examples to change W each epoch
- If $m = 1 \implies$ Stochastic Gradient Descent
  - We lose the parallelization of the calculation on multiple examples (and we have more error)
- If $1 < m <<< n \implies$ Mini-Batch Gradient Descent
  - We accelerate the update of the weights W and parallelize calculations on m examples

Gradient Descent

Let m be the size of a min-batch

- If $m = n \implies$ Batch Gradient Descent
  - We should calculate the error and gradient on all training examples to change W each epoch
- If $m = 1 \implies$ Stochastic Gradient Descent
  - We lose the parallelization of the calculation on multiple examples (and we have more error)
- If $1 < m <<< n \implies$ Mini-Batch Gradient Descent
  - We accelerate the update of the weights W and parallelize calculations on m examples

GD	Update On	Pros	Cons
Batch	Full dataset	Stable, global convergence	Computationally expensive
Stochastic	Single example	Fast convergence, escapes local minima	Noisy updates
Mini-Batch	Mini-batch subset	Balance of stability and speed	Computational overhead

Gradient Descent

https://twitter.com/docmilanfar/status/1755141174313308652?t=0Q6AErLOVIPBfj2smtszJQ

Linear Regression

Linear Regression - Cost Function

General form :

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} erreur(y_i, \hat{y}_i) \tag{8}$$

Linear Regression - Cost Function

General form :

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} erreur(y_i, \hat{y}_i) \tag{8}$$

Average quadratic error:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2 \tag{9}$$

Linear Regression - Cost Function

General form :

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} erreur(y_i, \hat{y}_i) \tag{8}$$

Average quadratic error:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2 \tag{9}$$

Average quadratic error for linear regression

$$L(w, b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w.x_i + b))^2 \tag{10}$$

Linear Regression - Cost Function

General form :

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} erreur(y_i, \hat{y}_i) \tag{8}$$

Average quadratic error:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2 \tag{9}$$

Average quadratic error for linear regression

$$L(w, b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w.x_i + b))^2 \tag{10}$$

For linear regression, two parameters to learn: w **and** b

Linear Regression - Calculate Gradient with respect to w

$$\frac{\partial L(w,b)}{\partial w} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial w} \tag{11}$$

Linear Regression - Calculate Gradient with respect to w

$$\frac{\partial L(w,b)}{\partial w} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial w} \tag{11}$$

$$\frac{\partial L(w,b)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial w} \tag{12}$$

Linear Regression - Calculate Gradient with respect to w

$$\frac{\partial L(w,b)}{\partial w} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial w} \tag{11}$$

$$\frac{\partial L(w,b)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial w} \tag{12}$$

however $\frac{\partial y_i}{\partial w} = 0$ because y_i is independent of w :

Linear Regression - Calculate Gradient with respect to w

$$\frac{\partial L(w,b)}{\partial w} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial w} \tag{11}$$

$$\frac{\partial L(w,b)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial w} \tag{12}$$

however $\frac{\partial y_i}{\partial w} = 0$ because y_i is independent of w :

$$\implies \frac{\partial L(w,b)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}(-2x_i(y_i - (w.x_i + b))) \tag{13}$$

Linear Regression - Calculate Gradient with respect to w

$$\frac{\partial L(w,b)}{\partial w} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial w} \tag{11}$$

$$\frac{\partial L(w,b)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial w} \tag{12}$$

however $\frac{\partial y_i}{\partial w} = 0$ because y_i is independent of w :

$$\implies \frac{\partial L(w,b)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}(-2x_i(y_i - (w.x_i + b))) \tag{13}$$

$$\implies \frac{\partial L(w,b)}{\partial w} = -\frac{2}{n}\sum_{i=1}^{n}(x_i(y_i - \hat{y}_i)) \tag{14}$$

Linear Regression - Calculate Gradient with respect to w

$$\frac{\partial L(w,b)}{\partial w} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial w} \tag{11}$$

$$\frac{\partial L(w,b)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial w} \tag{12}$$

however $\frac{\partial y_i}{\partial w} = 0$ because y_i is independent of w :

$$\implies \frac{\partial L(w,b)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}(-2x_i(y_i - (w.x_i + b))) \tag{13}$$

$$\implies \frac{\partial L(w,b)}{\partial w} = -\frac{2}{n}\sum_{i=1}^{n}(x_i(y_i - \hat{y}_i)) \tag{14}$$

Updating w :

$$w = w - \alpha.(-\frac{2}{n}\sum_{i=1}^{n}(x_i(y_i - \hat{y}_i))) \tag{15}$$

Linear Regression - Calculate Gradient with respect to b

$$\frac{\partial L(w,b)}{\partial b} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial b} \tag{16}$$

Linear Regression - Calculate Gradient with respect to b

$$\frac{\partial L(w,b)}{\partial b} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial b} \tag{16}$$

$$\frac{\partial L(w,b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial b} \tag{17}$$

Linear Regression - Calculate Gradient with respect to b

$$\frac{\partial L(w,b)}{\partial b} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial b} \tag{16}$$

$$\frac{\partial L(w,b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial b} \tag{17}$$

however $\frac{\partial y_i}{\partial b} = 0$ because y_i is independent of b :

Linear Regression - Calculate Gradient with respect to b

$$\frac{\partial L(w,b)}{\partial b} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial b} \tag{16}$$

$$\frac{\partial L(w,b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial b} \tag{17}$$

however $\frac{\partial y_i}{\partial b} = 0$ because y_i is independent of b :

$$\implies \frac{\partial L(w,b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(-2(y_i - (w.x_i + b))) \tag{18}$$

Linear Regression - Calculate Gradient with respect to b

$$\frac{\partial L(w,b)}{\partial b} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial b} \tag{16}$$

$$\frac{\partial L(w,b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial b} \tag{17}$$

however $\frac{\partial y_i}{\partial b} = 0$ because y_i is independent of b :

$$\implies \frac{\partial L(w,b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(-2(y_i - (w.x_i + b))) \tag{18}$$

$$\implies \frac{\partial L(w,b)}{\partial b} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{19}$$

Linear Regression - Calculate Gradient with respect to b

$$\frac{\partial L(w,b)}{\partial b} = \frac{\partial(\frac{1}{n}\sum_{i=1}^{n}(y_i - (w.x_i + b))^2)}{\partial b} \tag{16}$$

$$\frac{\partial L(w,b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial(y_i - (w.x_i + b))^2}{\partial b} \tag{17}$$

however $\frac{\partial y_i}{\partial b} = 0$ because y_i is independent of b :

$$\implies \frac{\partial L(w,b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(-2(y_i - (w.x_i + b))) \tag{18}$$

$$\implies \frac{\partial L(w,b)}{\partial b} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{19}$$

Updating b :

$$b = b - \alpha.(-\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)) \tag{20}$$

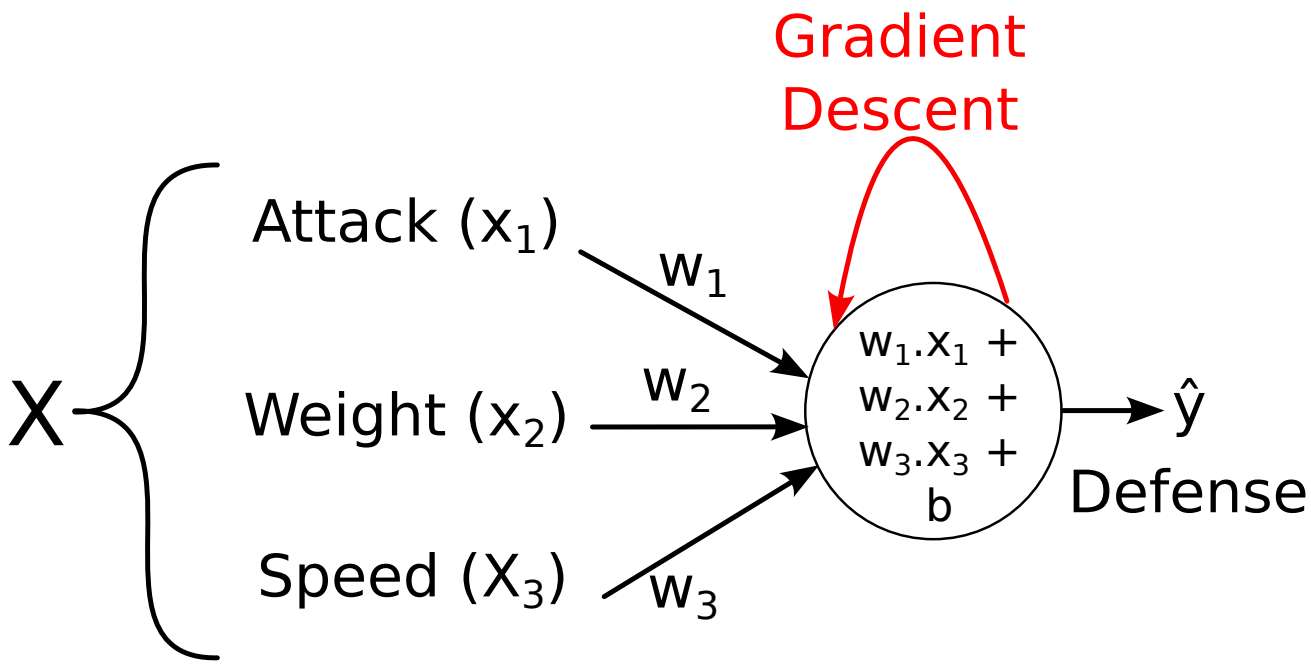
Linear Regression - With Many Variables

If the examples in our dataset have many attributes:

- Example: predict the defense value of a Pokemon in function of its attack value, weight and speed.
- Instead of having one parameter w, we will learn a set of parameters $W = (w_1, w_2, \ldots, w_d)$ for d characteristics in the data
- We repeat the same procedure for every $w_j \in W$
- The linear model becomes: $\hat{y}_i = b + \sum_{j=1}^{d} w_j . x_{i,j}$
- With $x_{i,j}$ being the j_{th} attribute of the i_{th} example in the data

Linear Regression - With Many Variables

Perceptron:



Linear Regression - Normalization of the Data

Normalizing the input data

- Involves scaling each feature to be between 0 and 1
- Its important that the features have the same distribution

Linear Regression - Normalization of the Data

Normalizing the input data

- Involves scaling each feature to be between 0 and 1
- Its important that the features have the same distribution
- For example:
  - the attack values of all Pokemon are between 10 and 194
  - the weight values of all Pokemon are between 0.1 and 999.9

Linear Regression - Normalization of the Data

Normalizing the input data

- Involves scaling each feature to be between 0 and 1
- Its important that the features have the same distribution
- For example:
  - the attack values of all Pokemon are between 10 and 194
  - the weight values of all Pokemon are between 0.1 and 999.9

Without normalization $\Longrightarrow$ a small variation in attack values does not have a high effect compared to the high values of weight $\Longrightarrow$ the model will find it difficult to learn something

Linear Regression - Normalization of the Data

Normalizing the input data

- Involves scaling each feature to be between 0 and 1
- Its important that the features have the same distribution
- For example:
  - the attack values of all Pokemon are between 10 and 194
  - the weight values of all Pokemon are between 0.1 and 999.9

Without normalization $\implies$ a small variation in attack values does not have a high effect compared to the high values of weight $\implies$ the model will find it difficult to learn something

$$X_{:,j} = \frac{X_{:,j} - min(X_{:,j})}{max(X_{:,j}) - min(X_{:,j})} \ \forall \ j \in [1, \ldots, d] \tag{21}$$

d being the number of attributes in the dataset

Predicting Defense Values for Pokemon

Predicting Defense Values for Pokemon

name	weight_kg	speed	sp_attack	sp_defense	type
Bulbasaur	6.9	45	65	65	grass
Charmander	8.5	65	60	50	fire
Butterfree	32	70	90	80	bug
Squirtle	9	43	50	64	water

Predicting Defense Values for Pokemon

name	weight_kg	speed	sp_attack	sp_defense	type
Bulbasaur	6.9	45	65	65	grass
Charmander	8.5	65	60	50	fire
Butterfree	32	70	90	80	bug
Squirtle	9	43	50	64	water

- Each Pokemon has 6 total attributes (4 numerical and 2 categorical). 1 of the attributes is the value to predict, the other 5 attributes are to be used to teach the model
- One Pokemon is represented by a vector $x_i = (x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}, x_{i,5})$

Predicting Defense Values for Pokemon

name	weight_kg	speed	sp_attack	sp_defense	type
Bulbasaur	6.9	45	65	65	grass
Charmander	8.5	65	60	50	fire
Butterfree	32	70	90	80	bug
Squirtle	9	43	50	64	water

- Each Pokemon has 6 total attributes (4 numerical and 2 categorical). 1 of the attributes is the value to predict, the other 5 attributes are to be used to teach the model
- One Pokemon is represented by a vector $x_i = (x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}, x_{i,5})$
- 5 attributes $\implies$ 5 w_i and 1 b
- Let be a vector $W = (w_1, w_2, w_3, w_4, w_5)$

Predicting Defense Values for Pokemon

name	weight_kg	speed	sp_attack	sp_defense	type
Bulbasaur	6.9	45	65	65	grass
Charmander	8.5	65	60	50	fire
Butterfree	32	70	90	80	bug
Squirtle	9	43	50	64	water

Predicting Defense Values for Pokemon

name	weight_kg	speed	sp_attack	sp_defense	type
Bulbasaur	6.9	45	65	65	grass
Charmander	8.5	65	60	50	fire
Butterfree	32	70	90	80	bug
Squirtle	9	43	50	64	water

Data that is of interest to us:

weight_kg	sp_attack	sp_defense
6.9	65	65
8.5	60	50
32	90	80
9	50	64

- Attributes used to teach the model: weight_kg and sp_attack
- Attribute to predict: sp_defense

Predicting Defense Values for Pokemon

Goal: predict the defense value of a Pokemon in function of 2 attributes

$$defense = f(X) \mid X = [X_{:,1}, X_{:,2}] \tag{22}$$

Predicting Defense Values for Pokemon

Goal: predict the defense value of a Pokemon in function of 2 attributes

$$defense = f(X) \mid X = [X_{:,1}, X_{:,2}] \tag{22}$$

Model to learn

$$\hat{y} = W.X + b \equiv \hat{y} = w_1.X_{:,1} + w_2.X_{:,2} + b \tag{23}$$

Predicting Defense Values for Pokemon

Goal: predict the defense value of a Pokemon in function of 2 attributes

$$defense = f(X) \mid X = [X_{:,1}, X_{:,2}] \tag{22}$$

Model to learn

$$\hat{y} = W.X + b \equiv \hat{y} = w_1.X_{:,1} + w_2.X_{:,2} + b \tag{23}$$

This equation corresponds to a plane in the 3D space

$$z = w_1.x + w_2.y + b \tag{24}$$

Predicting Defense Values for Pokemon

Goal: predict the defense value of a Pokemon in function of 2 attributes

$$defense = f(X) \mid X = [X_{:,1}, X_{:,2}] \tag{22}$$

Model to learn

$$\hat{y} = W.X + b \equiv \hat{y} = w_1.X_{:,1} + w_2.X_{:,2} + b \tag{23}$$

This equation corresponds to a plane in the 3D space

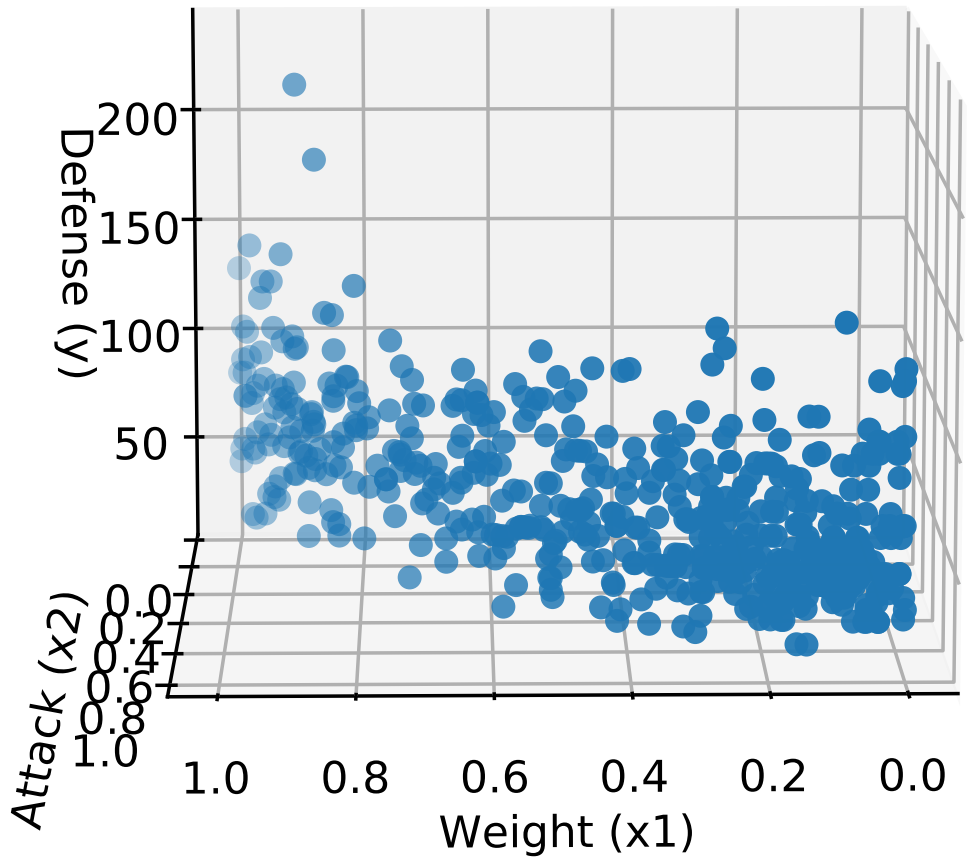
$$z = w_1.x + w_2.y + b \tag{24}$$

with

$$x = X_{:,1}; y = X_{:,2}; z = \hat{y} \tag{25}$$

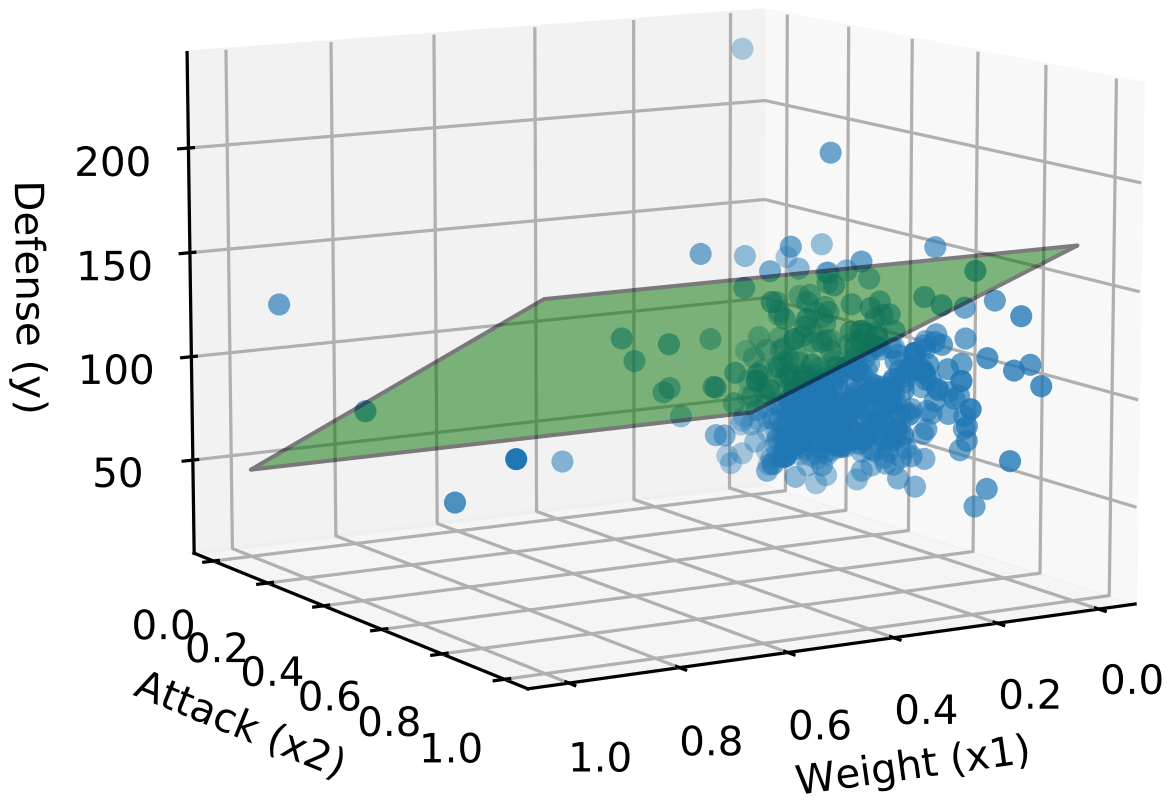
Predicting Defense Values for Pokemon

3D plot of the data



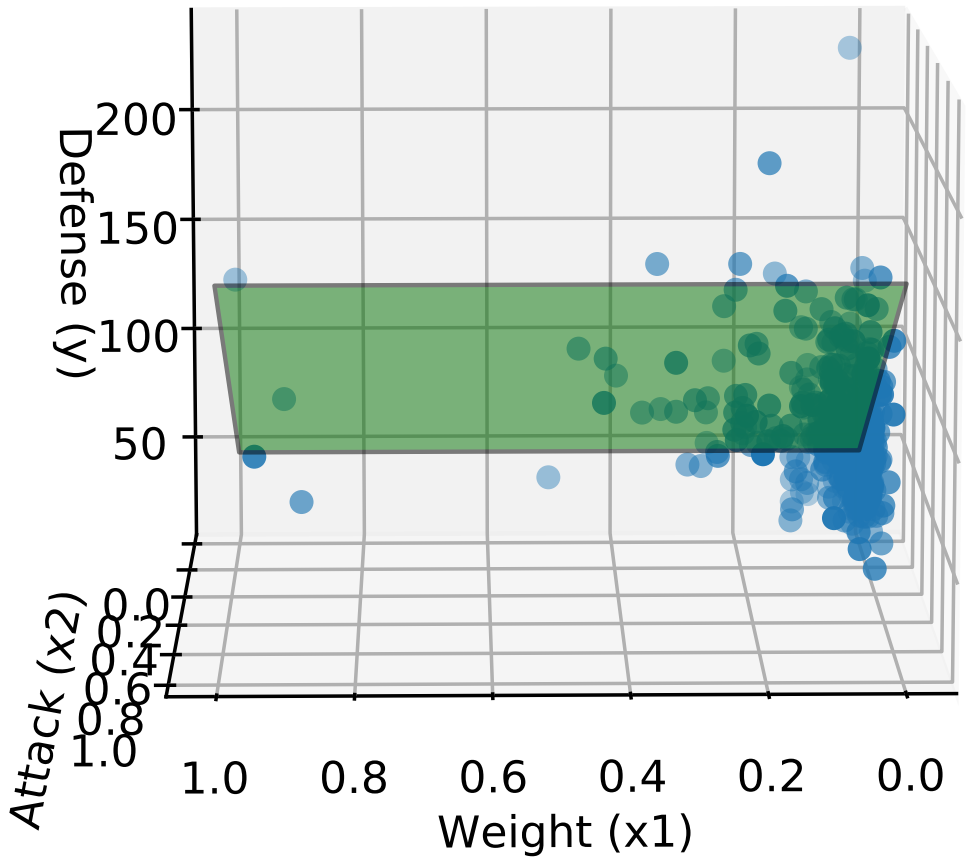
Predicting Defense Values for Pokemon

3D plot of data with decision plane



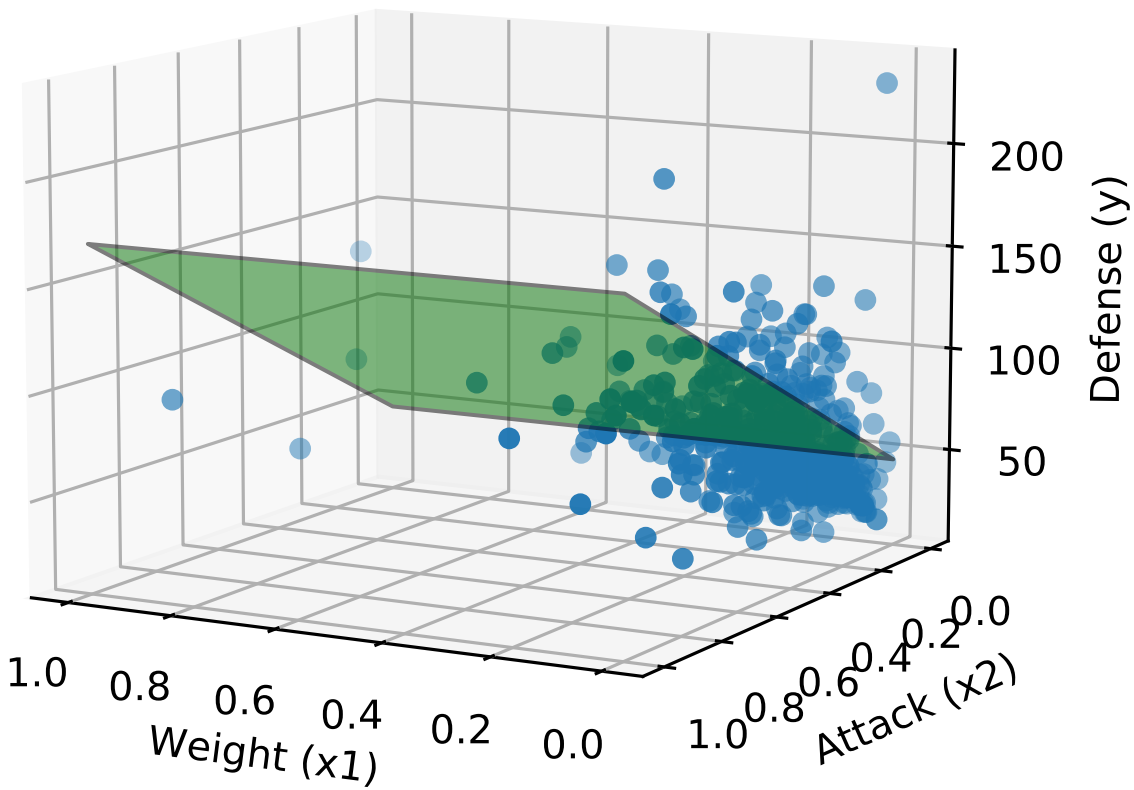
Predicting Defense Values for Pokemon

3D plot of data with decision plane



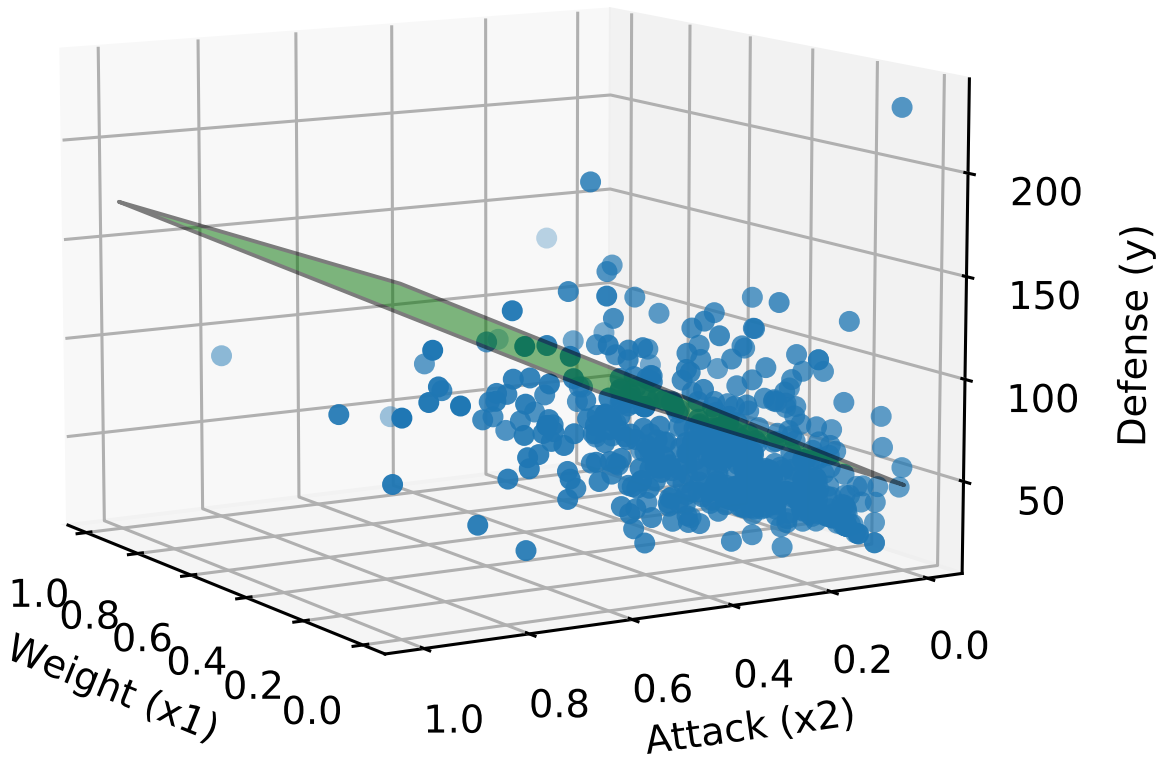
Predicting Defense Values for Pokemon

3D plot of data with decision plane



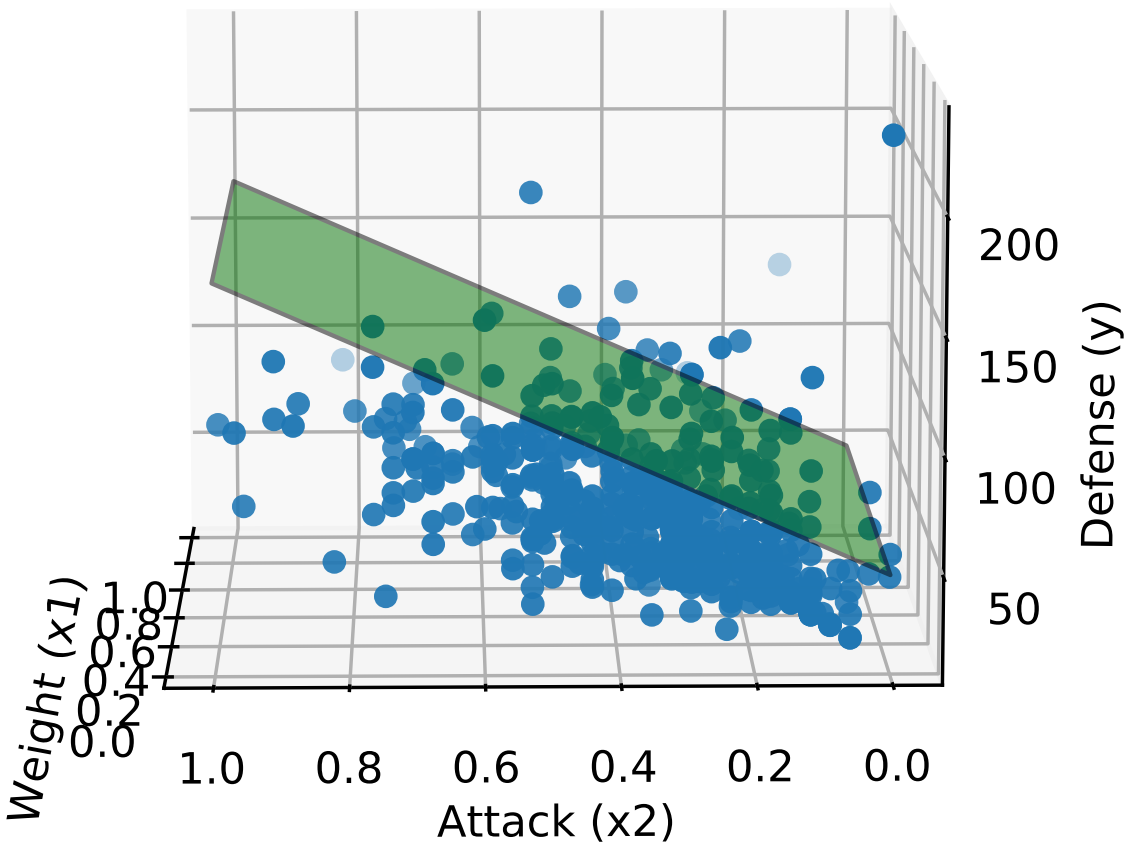
Predicting Defense Values for Pokemon

3D plot of data with decision plane



Predicting Defense Values for Pokemon

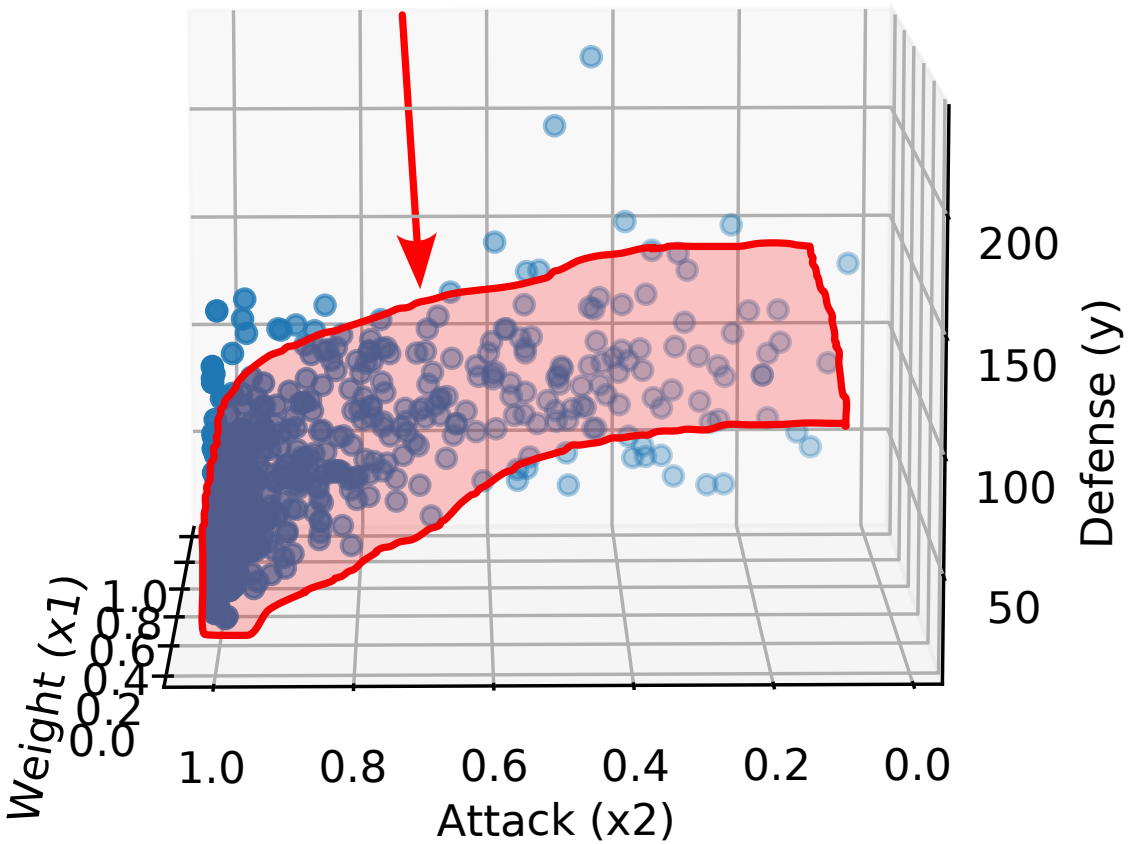
3D plot of data with decision plane



Predicting Defense Values for Pokemon

Is linear decision the best approach ?

seems to be a curved plane
so non-linear decision



Predicting Defense Values for Pokemon

- Non-Linear $\Longrightarrow$ non-linear interaction between attributes
- Example: x_1^2; $x_1.x_2$; $x_1^2 2 + x_2$ etc.

Predicting Defense Values for Pokemon

- Non-Linear $\Longrightarrow$ non-linear interaction between attributes
- Example: x_1^2; $x_1.x_2$; $x_1^2 2 + x_2$ etc.

$$\hat{y} = w_1.X_{:,1} + w_2.X_{:,2} + w_3.X_{:,3} + b \tag{26}$$

with:

$$X_{:,3} = X_{:,1}.X_{:,2} \tag{27}$$

Predicting Defense Values for Pokemon

- Non-Linear $\Longrightarrow$ non-linear interaction between attributes
- Example: x_1^2; $x_1.x_2$; $x_1^2 2 + x_2$ etc.

$$\hat{y} = w_1.X_{:,1} + w_2.X_{:,2} + w_3.X_{:,3} + b \tag{26}$$

with:

$$X_{:,3} = X_{:,1}.X_{:,2} \tag{27}$$

weight_kg	sp_attack	weight_kg . sp_attack	sp_defense
6.9	65	448,5	65
8.5	60	510	50
32	90	2880	80
9	50	450	64