Gradient Descent

Research Skills: Machine Learning

Grzegorz Chrupała
g.chrupala @ uvt.nl

Two aspects of a learner

- How it uses inputs to predict outputs
  - DT – traverse tree asking questions until arrive at leaf with **target**
  - Perceptron – predict **+1** if $\mathbf{w} \cdot \mathbf{x} + b <= 0$
- How it learns
  - DT – split data using best question and recurse on splits
  - Perceptron – update **w** and b if made a mistake

Two aspects of a learner

- How it uses inputs to predict outputs
  - **Model**
- How it learns
  - **Optimization algorithm**

Modularity

- Often **parameters** (e.g. weights) of the same model can be found in many different ways
- Standard **optimization** algorithms for many types of problems
  - Often can be treated as a **black box**

Linear models

- Linear models are based on a weighted sum of features
- (Multiclass) Classification
- (Multivariate) Regression

For example linear regression

$$y = \mathbf{w} \cdot \mathbf{x} + b$$

How can we find best w?

- Use specialized formula for linear regression

OR

- Convert into problem of finding minimum of function
- Standard solvers
  - Newton's method
  - (Stochastic) gradient descent
- **This approach works for many types of models**

Sum of squared errors

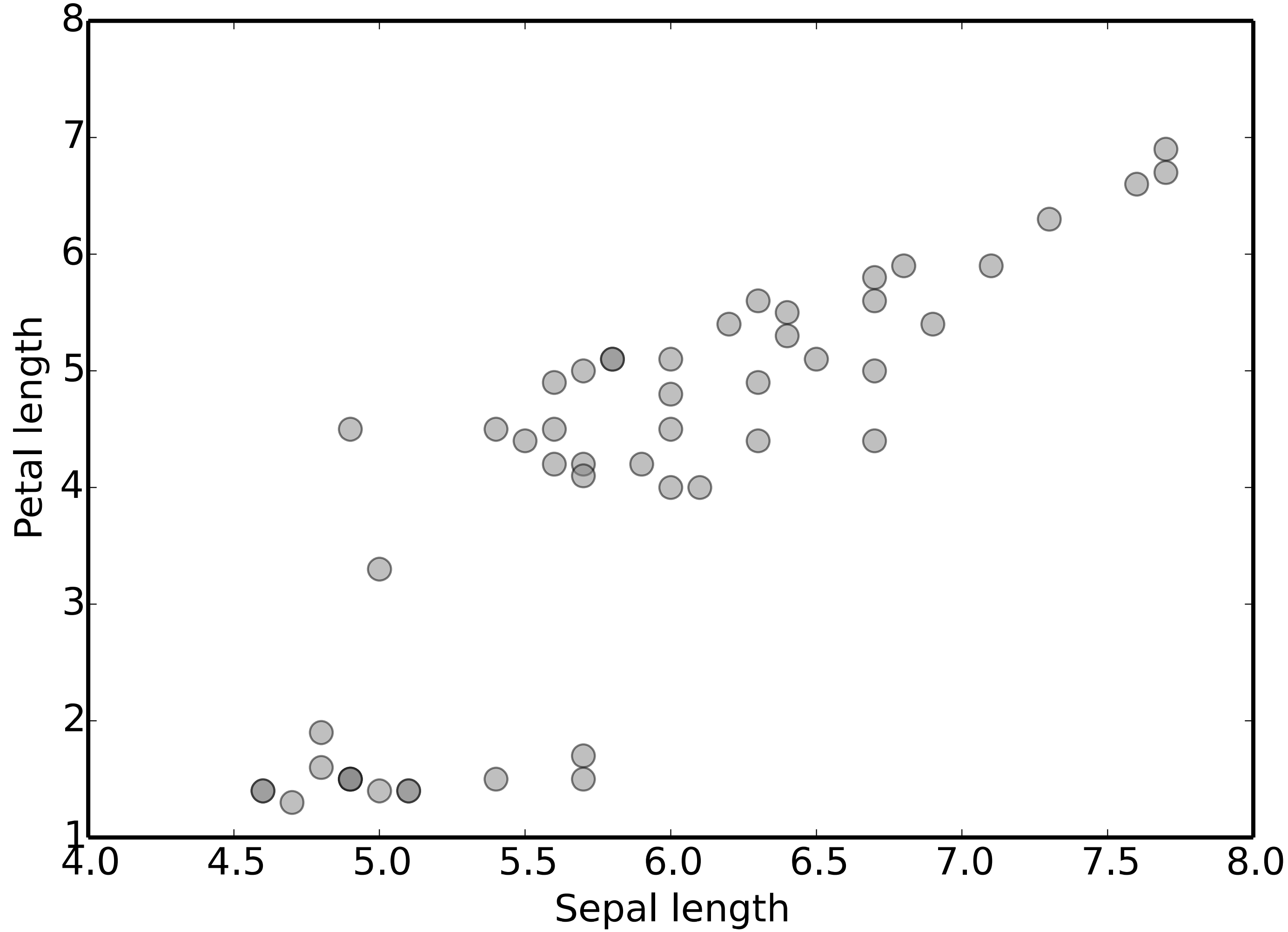
- Want to find **w**,b for which error on training data is smallest
- We can use SSE as a measure of error

$$\mathrm{SSE} = \sum_{i=1}^{N} (y^i_{\mathrm{pred}} - y^i)^2$$

Error as a function of w,b

$$\mathrm{Error}(\mathbf{w}, b) = \sum_{i=1}^{N} (\mathbf{w} \cdot \mathbf{x}^i + b - y^i)^2$$

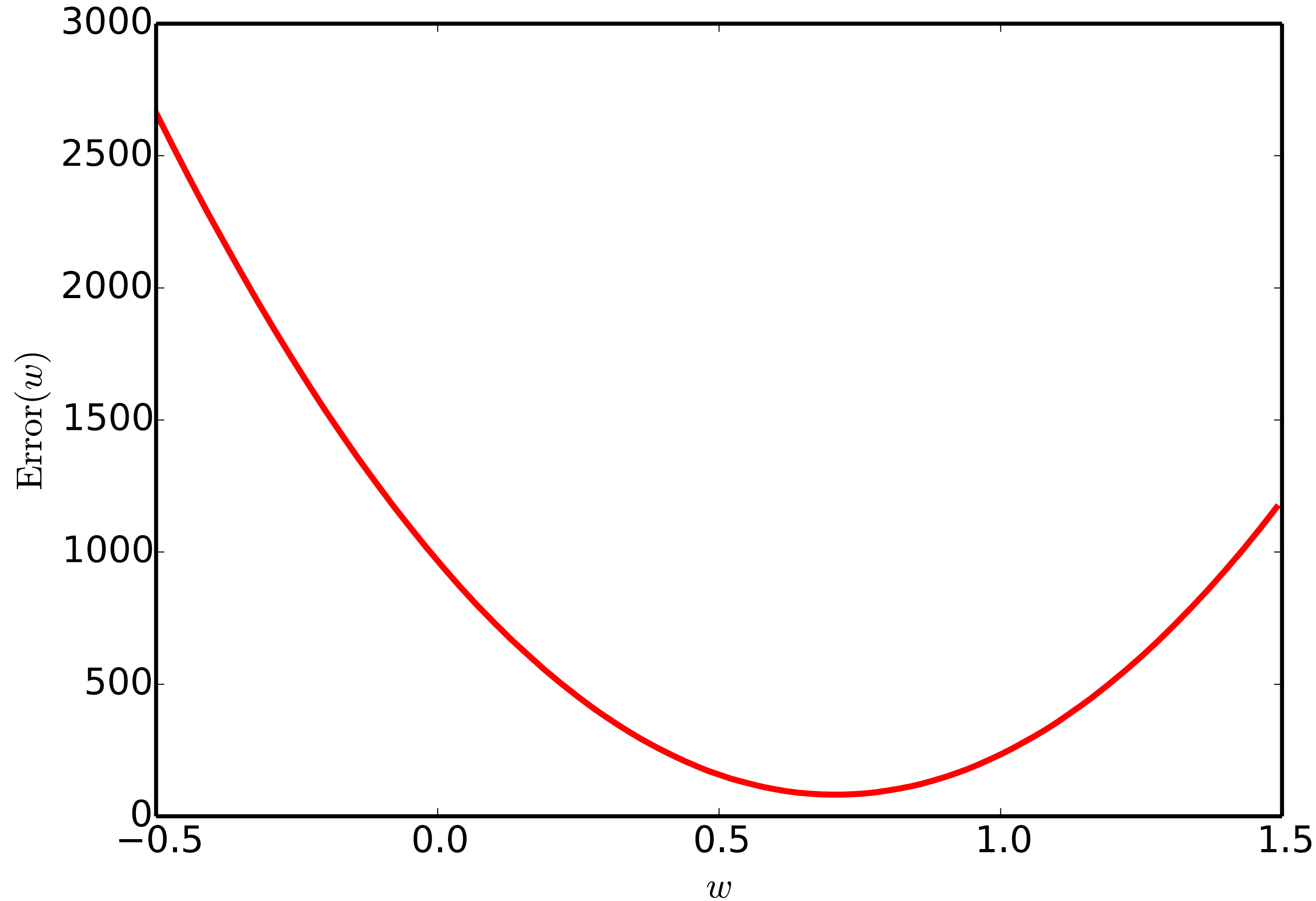
Example – Iris



Iris

- Find regression line which predicts **petal length** from **sepal length**
- For simplicity, fix $b = 0$
- How does **Error(w)** change as we vary w?

Find w for which Error(w) is lowest

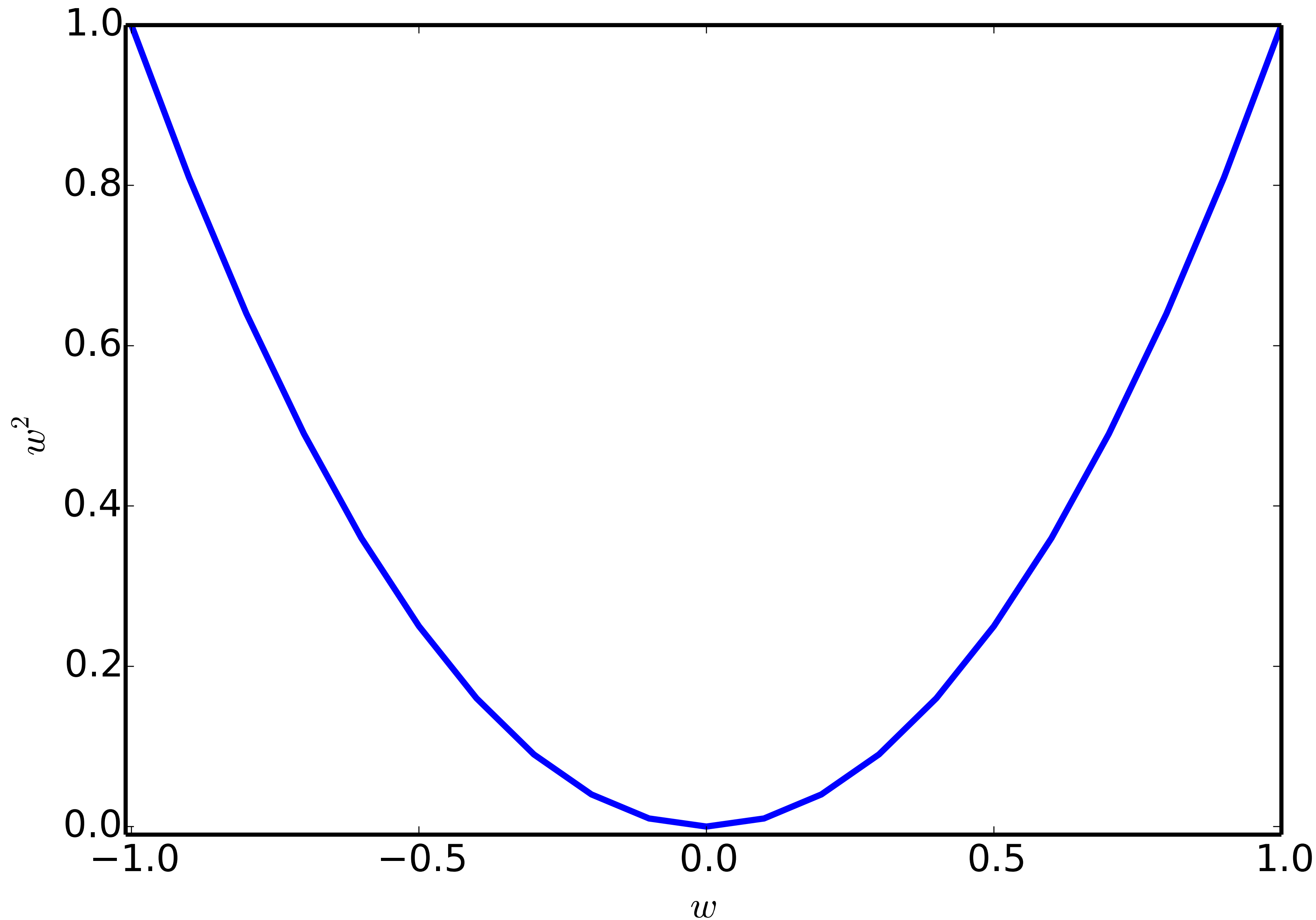


Start with something even simpler

$$\text{Error}(\mathbf{w}, b) = \sum_{i=1}^{N} (\mathbf{w} \cdot \mathbf{x}^i + b - y^i)^2$$

- Work through example of a simpler function:

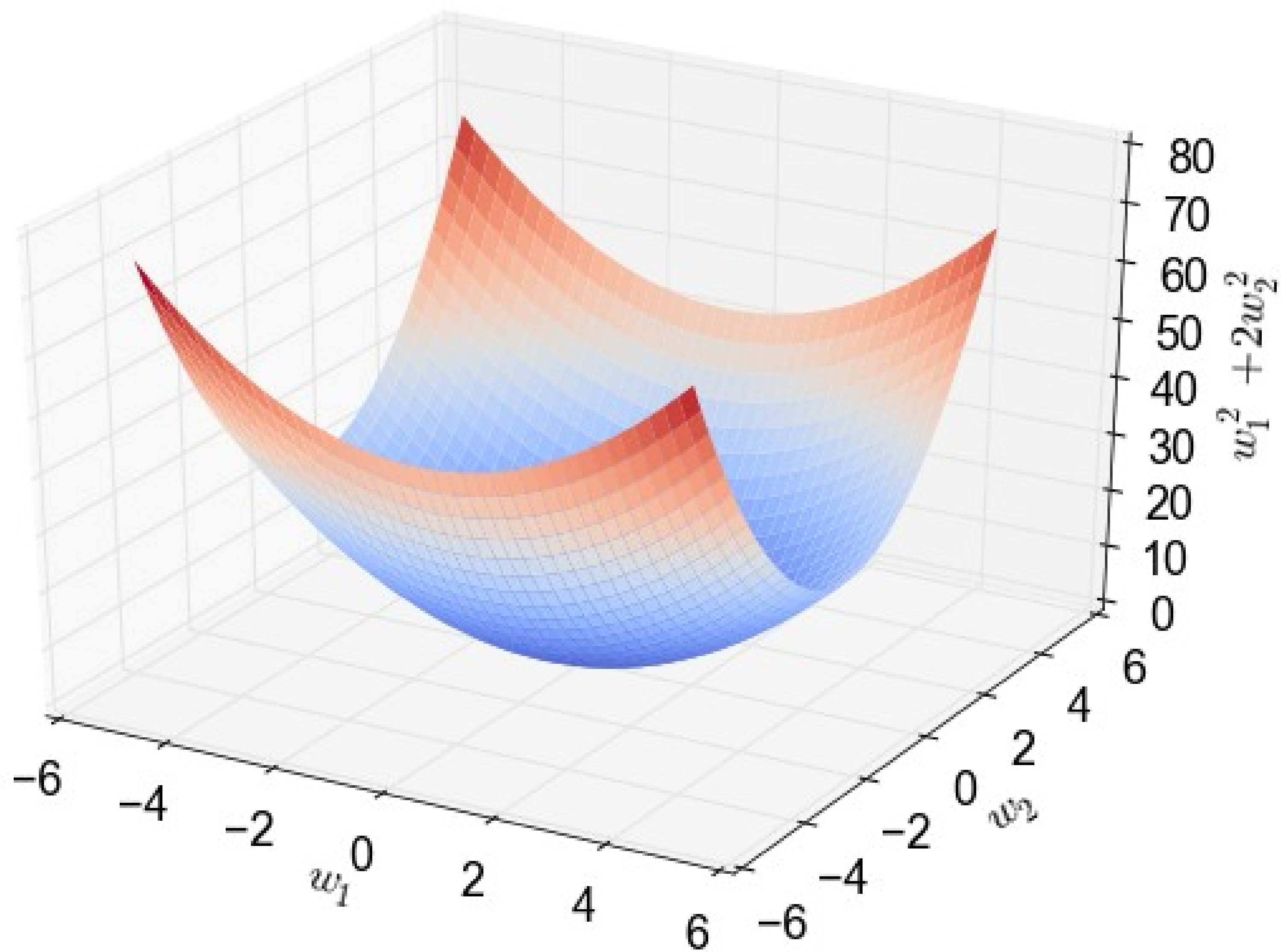
$$f(w) = w^2$$


1.0
0.8
0.6
0.4
0.2
0.0
w^2
−1.0
−0.5
0.0
0.5
1.0
w

- How can we find the value of w which minimizes $f(w)$?
- Start at a random value of w
- Check slope of function at this point
- Descend the slope: adjust w to decrease $f(w)$

Gradient vs slope

- **Slope** describes steepness of a single dimension
- We usually work with functions with vectors as arguments , e.g. Error(**w**)
- **Gradient** is the collection of slopes, one for each dimension



80
70
60
50
40
30
20
10
0
$w_1^2 + 2w_2^2$
−6
−4
−2
0
2
4
6
w_1
6
4
2
0
−2
−4
−6
w_2

Gradient descent for $f(w) = w^2$

How do we compute the slope of a function?

- First derivative!
- For function *f*, first derivative can be written *f'*
- Then *f'(a)* is the slope of function f at point *a*

First derivative

- If we define

$$f(w) = w^2$$

- The first derivative is

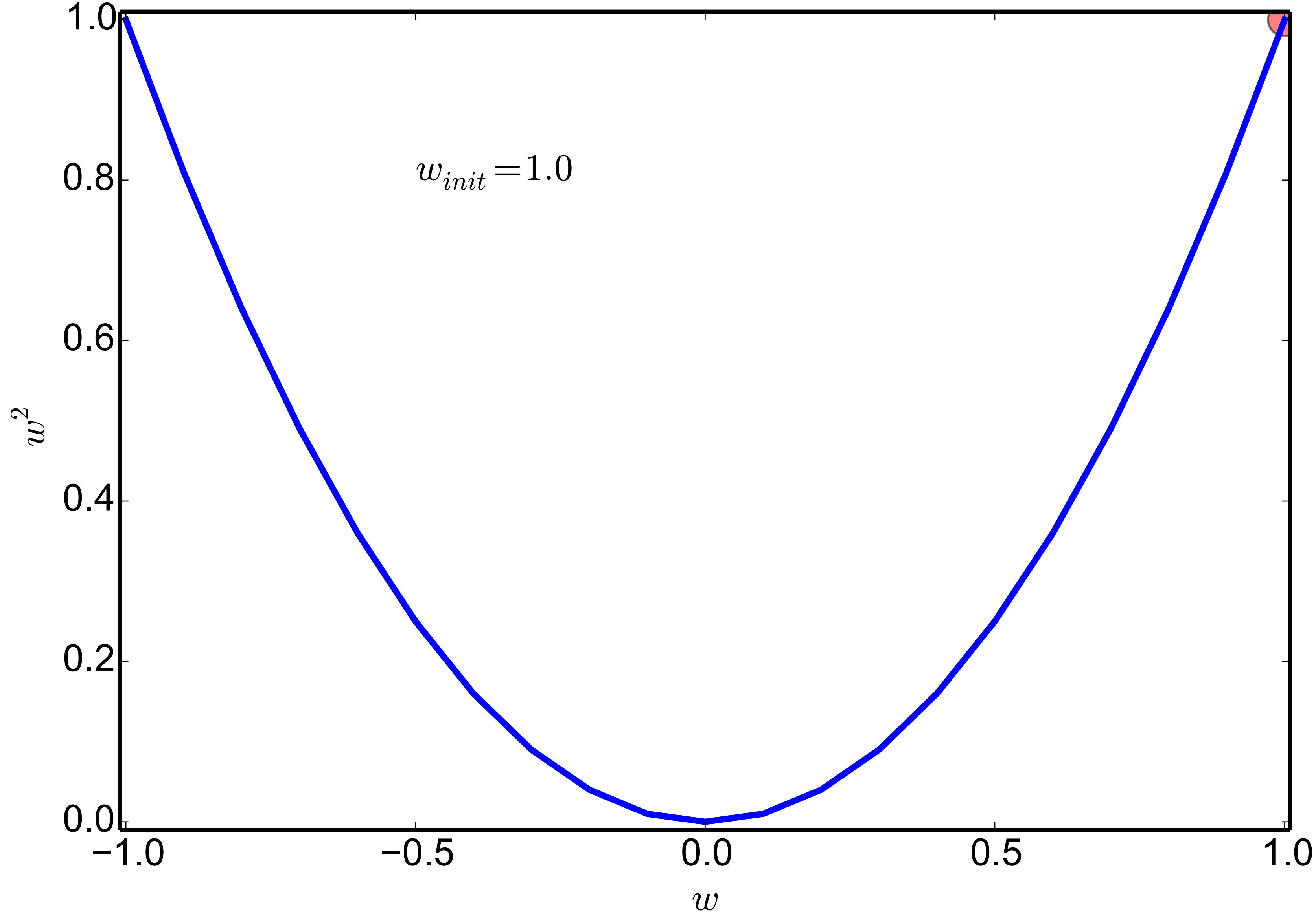
$$f'(w) = 2w$$

Ready to descend

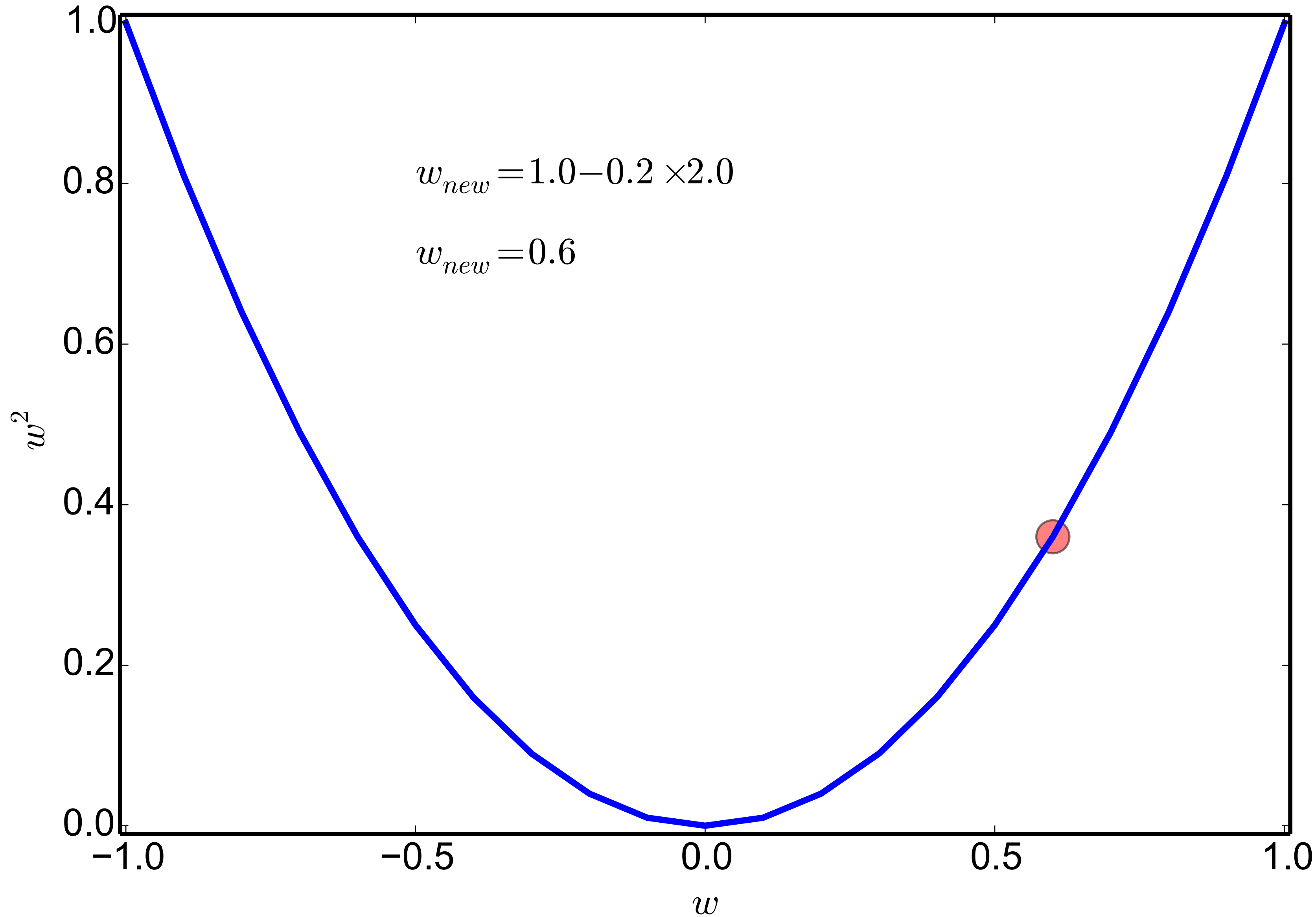
- Initialize w to some value (e.g. 1.0)
- Update:

$$w_{\mathrm{new}} = w_{\mathrm{old}} - \eta \times f'(w_{\mathrm{old}})$$

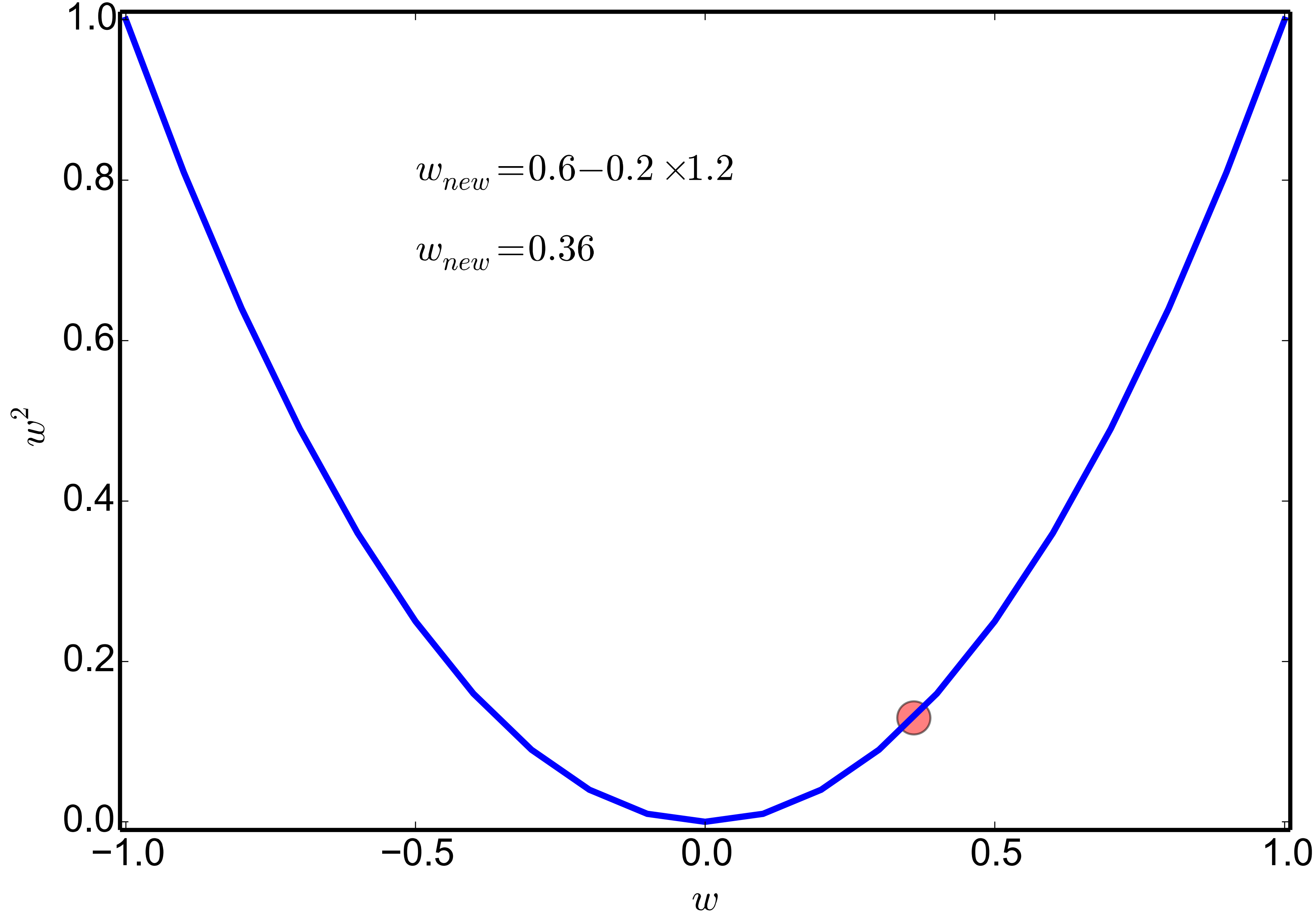
- η is the **learning rate**, controling speed of descent (e.g. 0.01 or 0.2)
- Stop when w doesn't change much any more



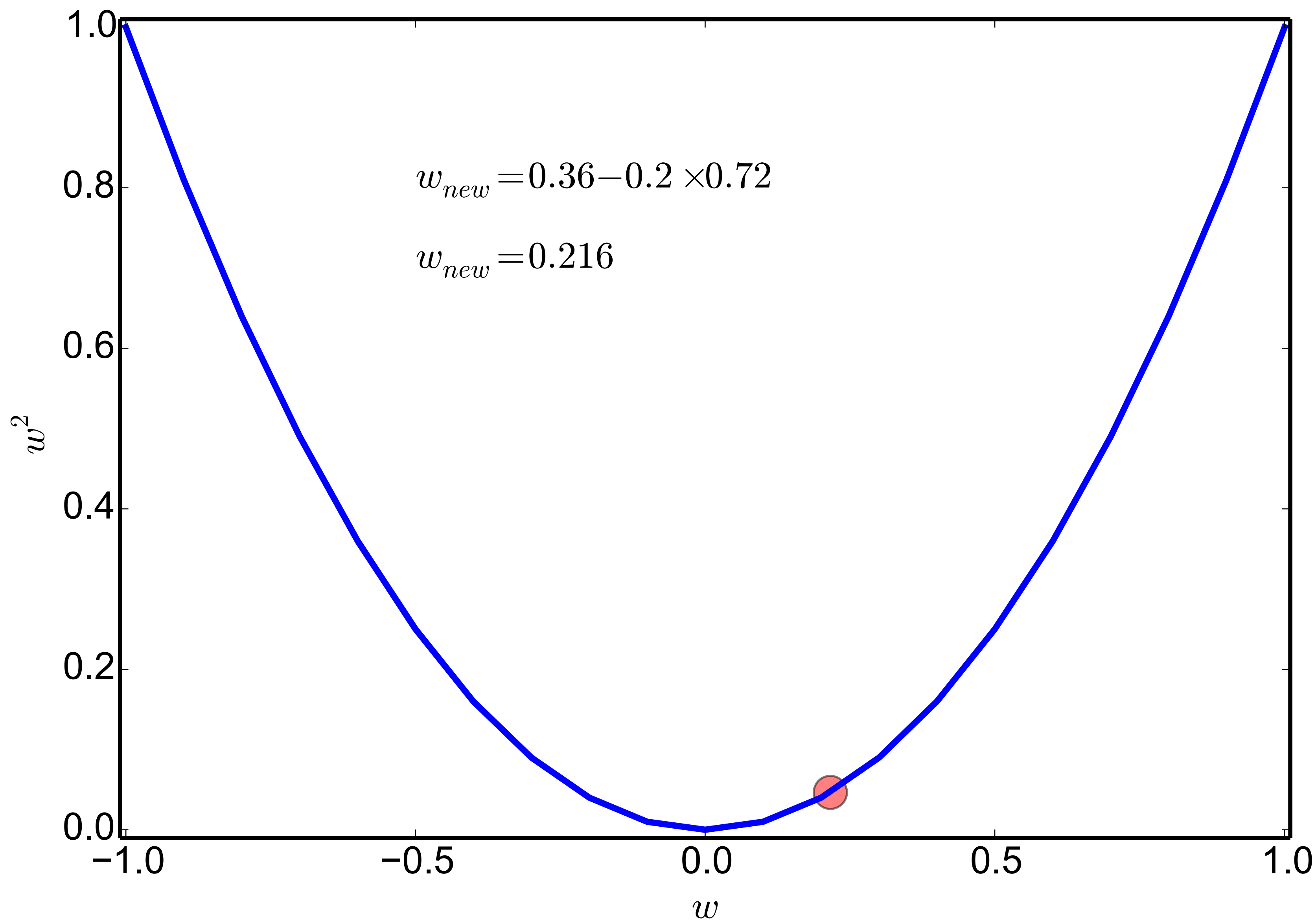
1.0
0.8
0.6
0.4
0.2
0.0
w^2
$w_{init} = 1.0$
−1.0
−0.5
0.0
0.5
1.0
w



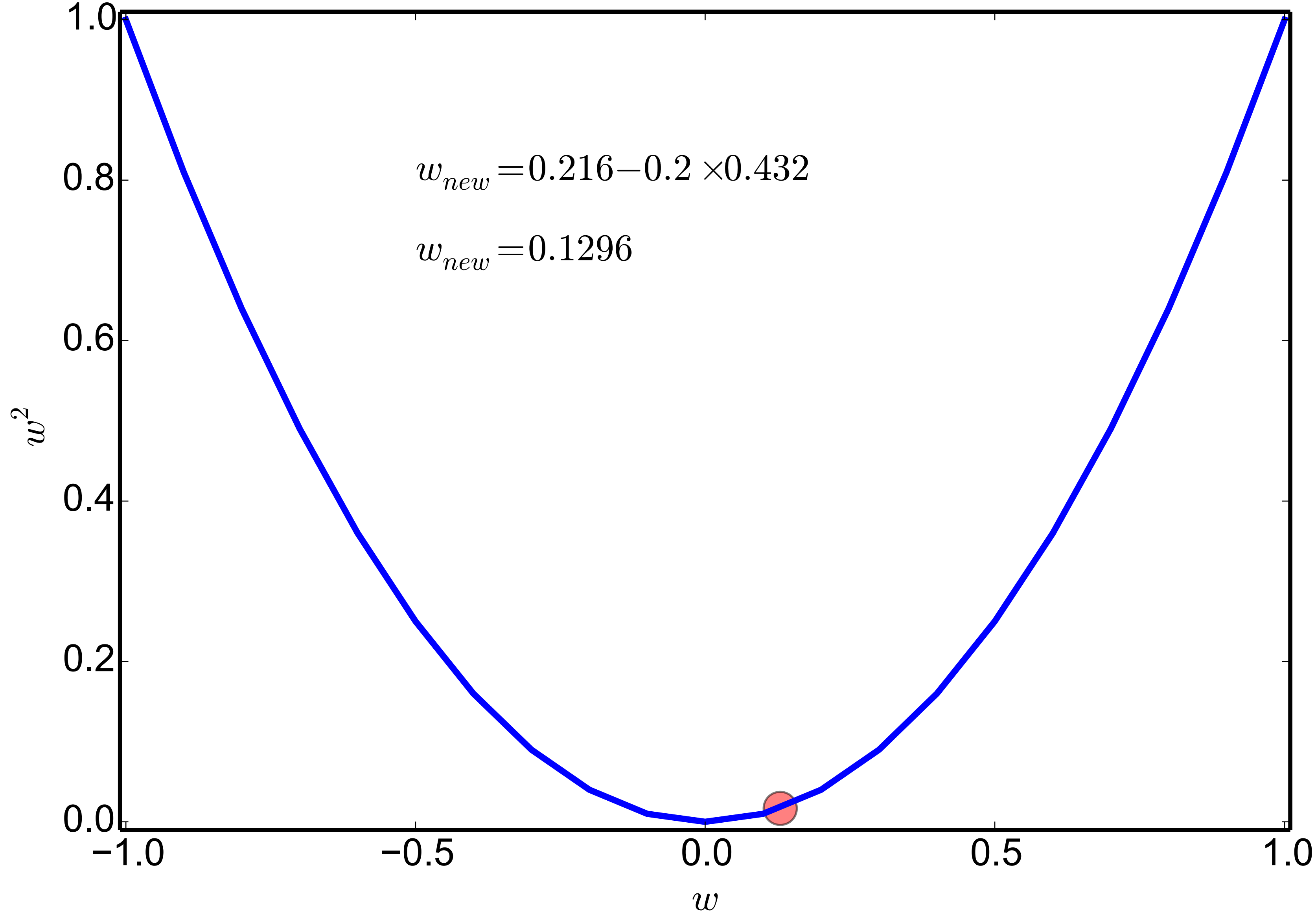
$w_{new} = 1.0 - 0.2 \times 2.0$
$w_{new} = 0.6$
w^2
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0
w

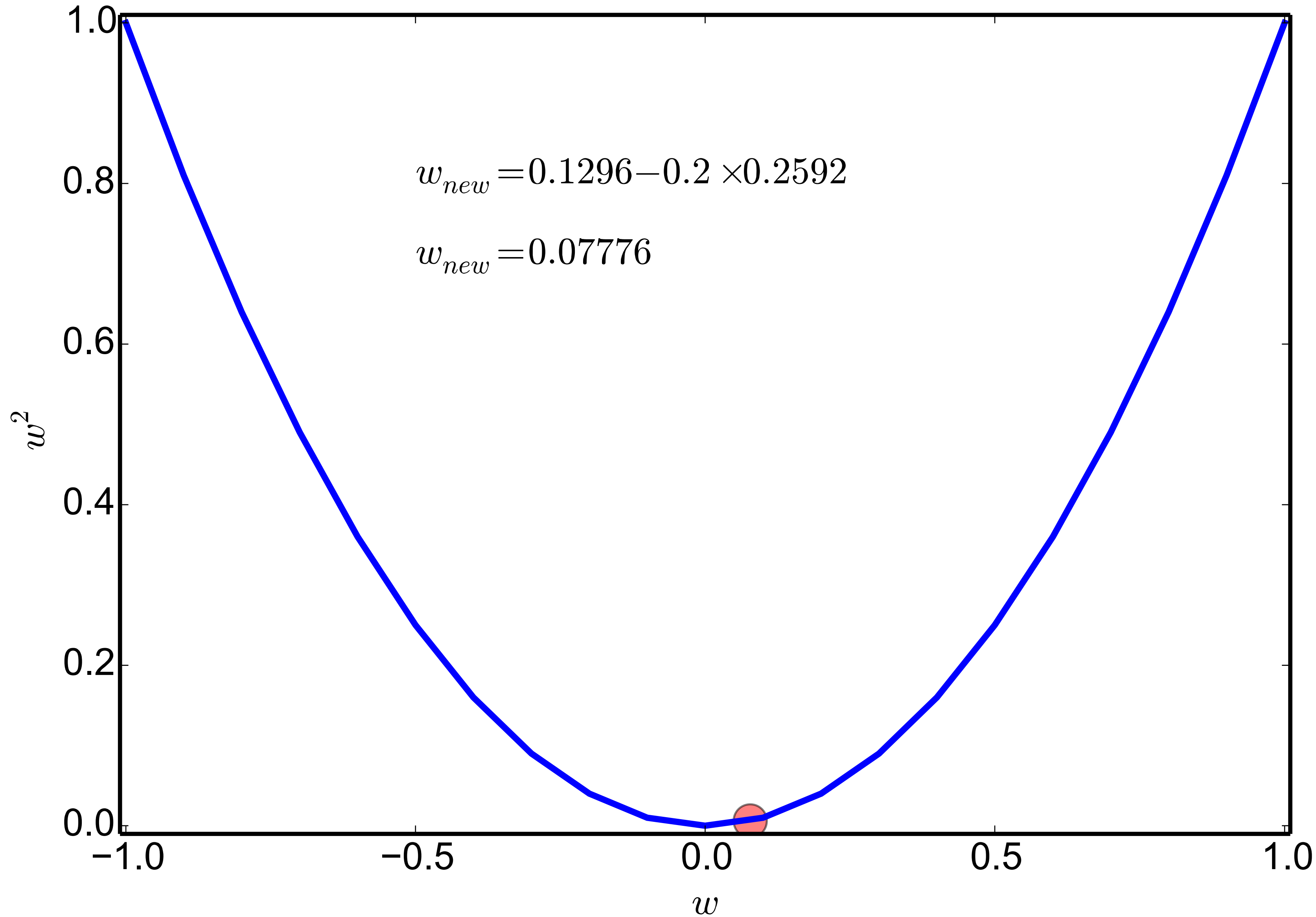


$w_{new} = 0.6 - 0.2 \times 1.2$
$w_{new} = 0.36$
w^2
w
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0



$w_{new} = 0.36 - 0.2 \times 0.72$
$w_{new} = 0.216$
1.0
0.8
0.6
0.4
0.2
0.0
w^2
−1.0
−0.5
0.0
0.5
1.0
w


$w_{new} = 0.216 - 0.2 \times 0.432$
$w_{new} = 0.1296$
1.0
0.8
0.6
0.4
0.2
0.0
w^2
−1.0
−0.5
0.0
0.5
1.0
w


$w_{new} = 0.1296 - 0.2 \times 0.2592$
$w_{new} = 0.07776$
1.0
0.8
0.6
0.4
0.2
0.0
w^2
−1.0
−0.5
0.0
0.5
1.0
w

Back to function Error(**w**,b)

$$b_{\text{new}} = b_{\text{old}} - \eta \times 2 \sum_{i=1}^{N} (y^i_{\text{pred}} - y^i)$$

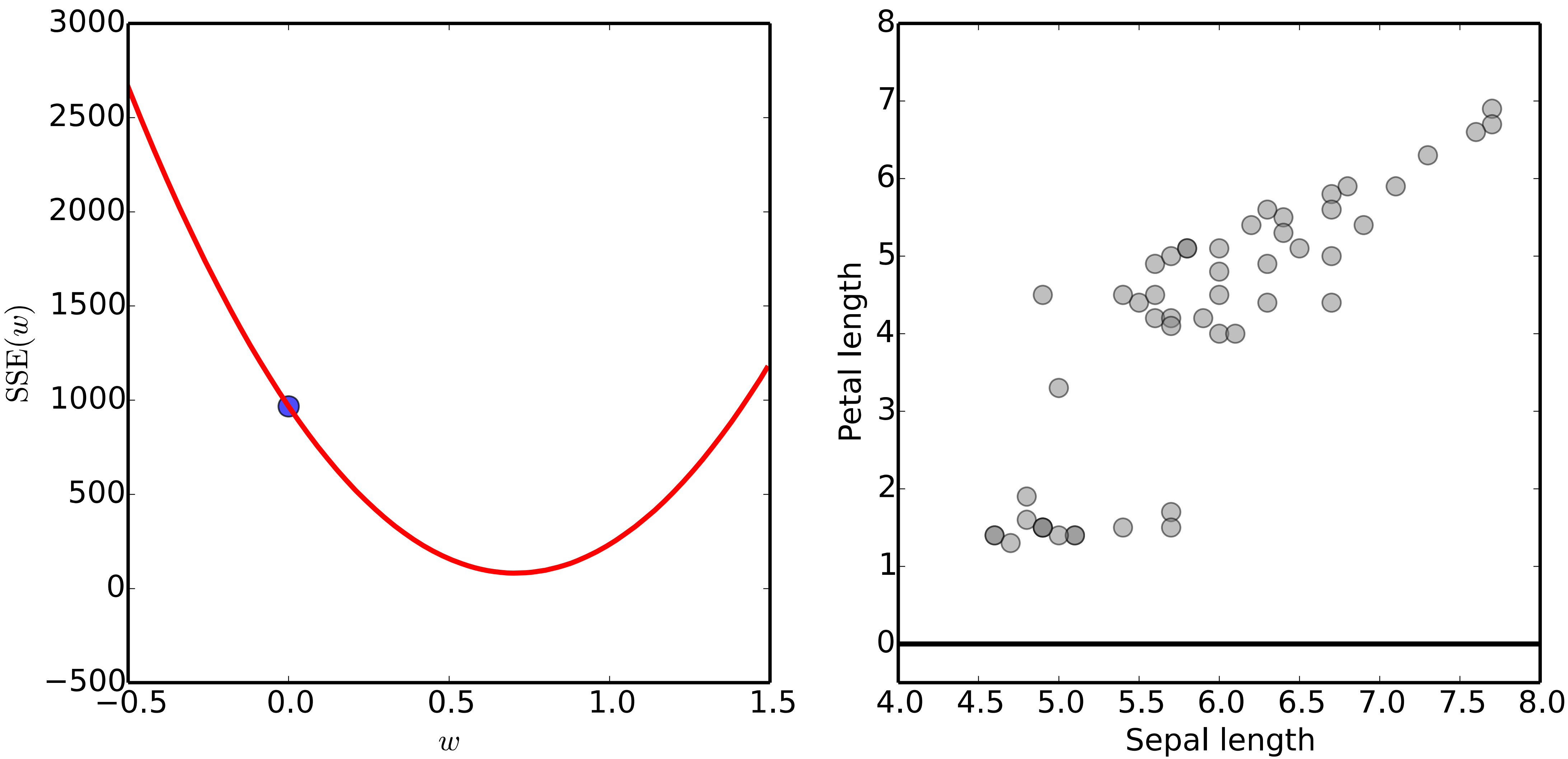
$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \eta \times 2 \sum_{i=1}^{N} (y^i_{\text{pred}} - y^i)\mathbf{x}^i$$

No need to memorize these formulas.

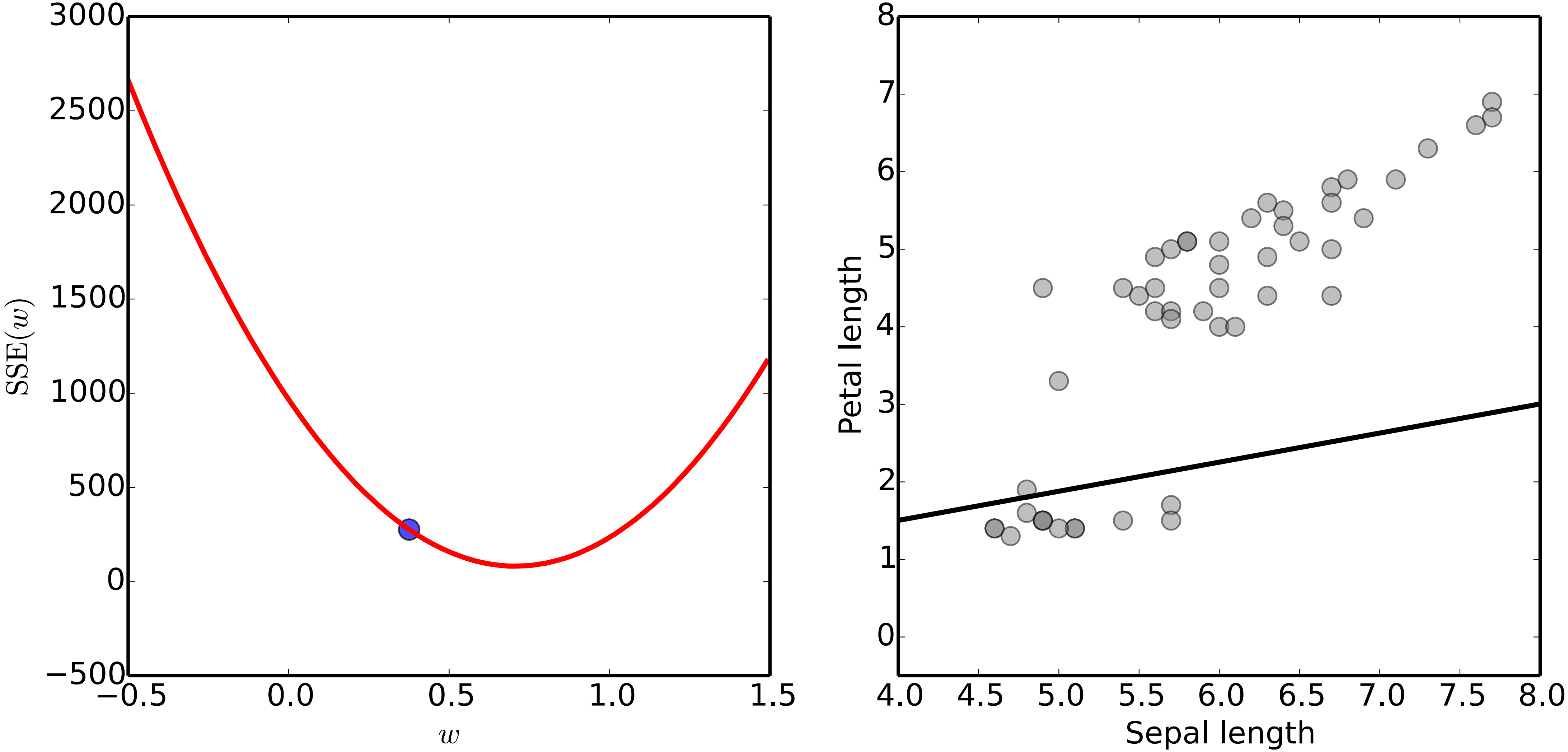
They are given only for reference.

Visualization

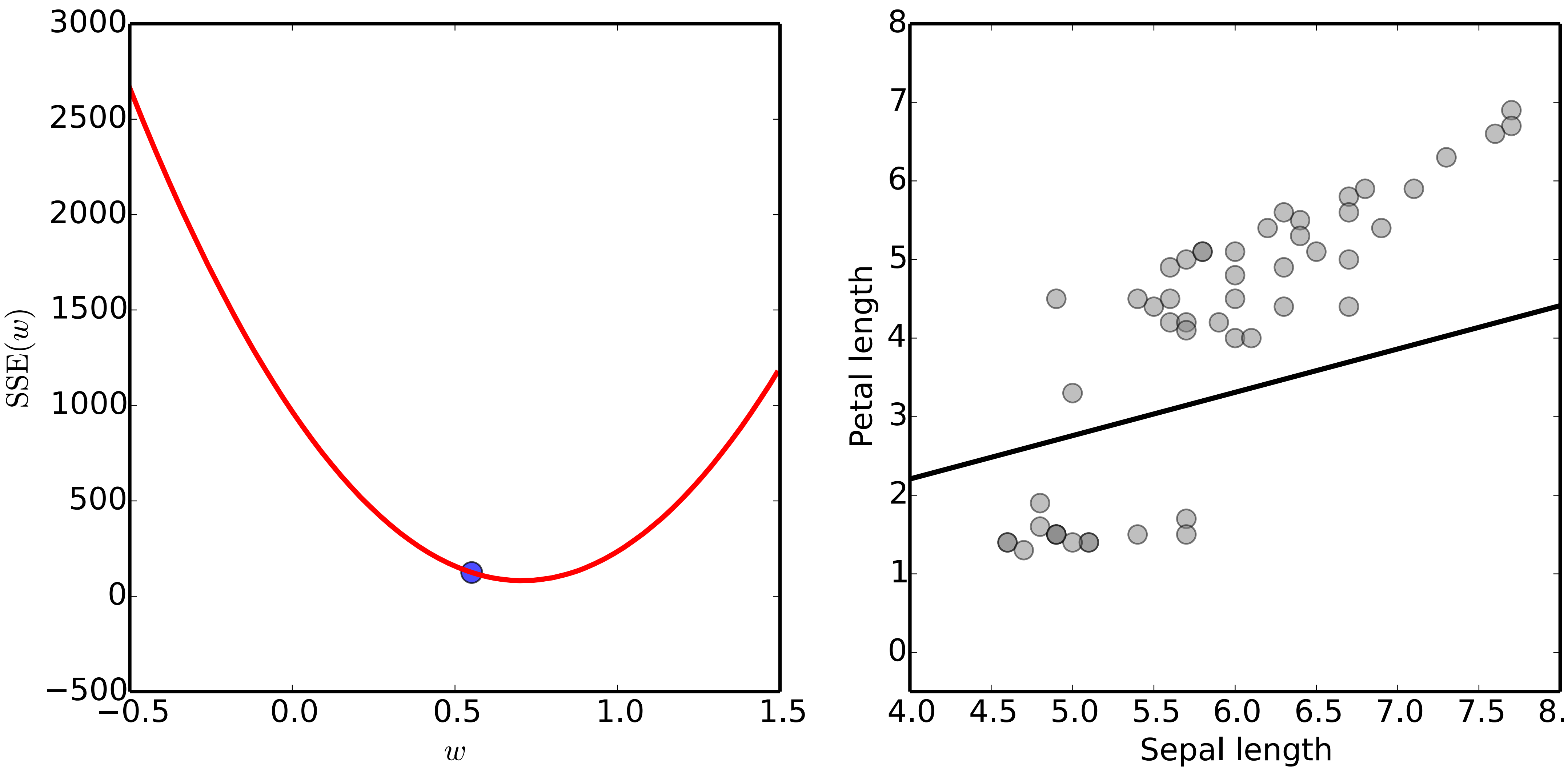
(Keep fixed b=0 to make it easier to plot)



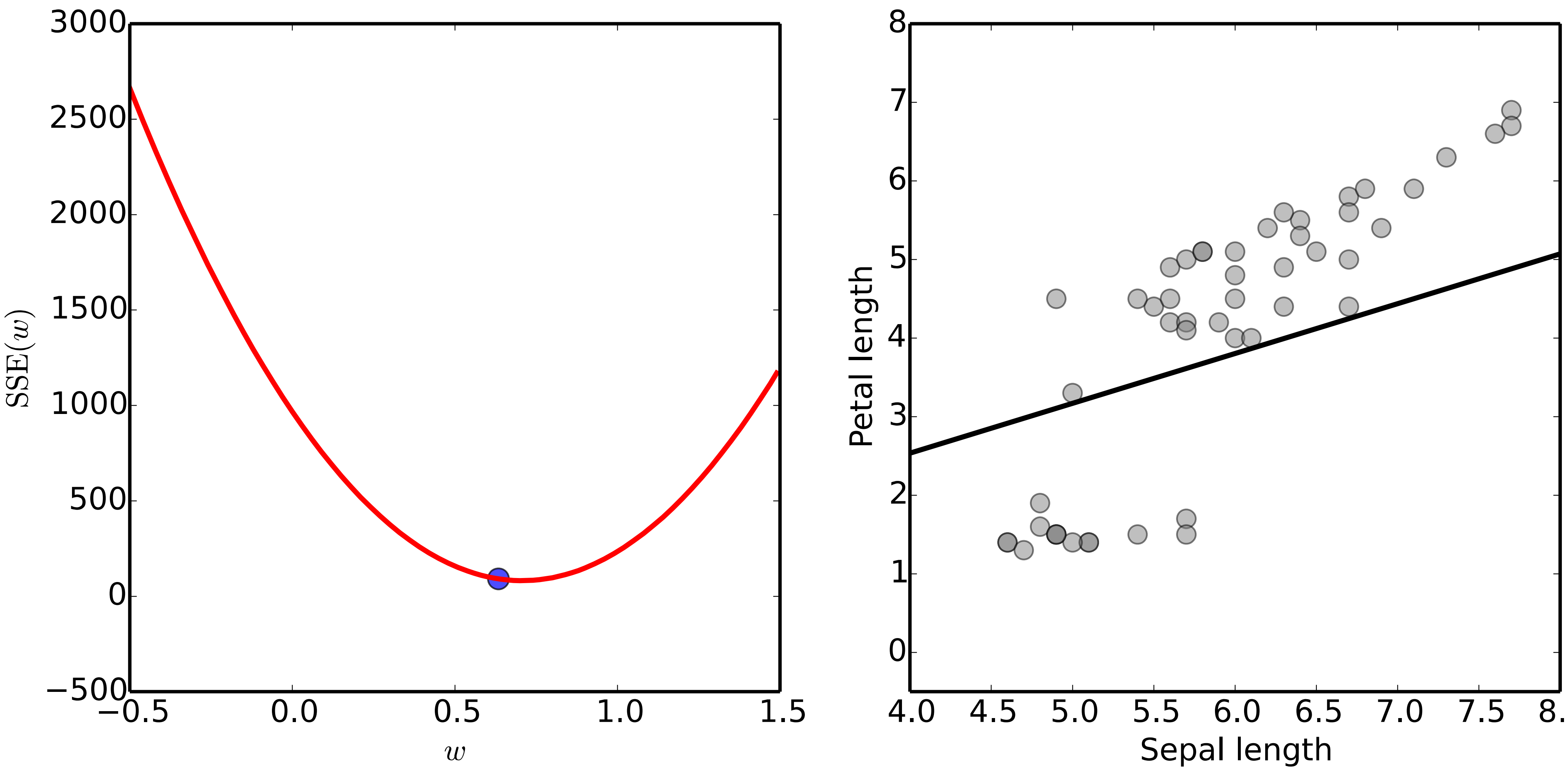
3000
2500
2000
1500
1000
500
0
−500
−0.5
0.0
0.5
1.0
1.5
SSE(w)
w
8
7
6
5
4
3
2
1
0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Petal length
Sepal length



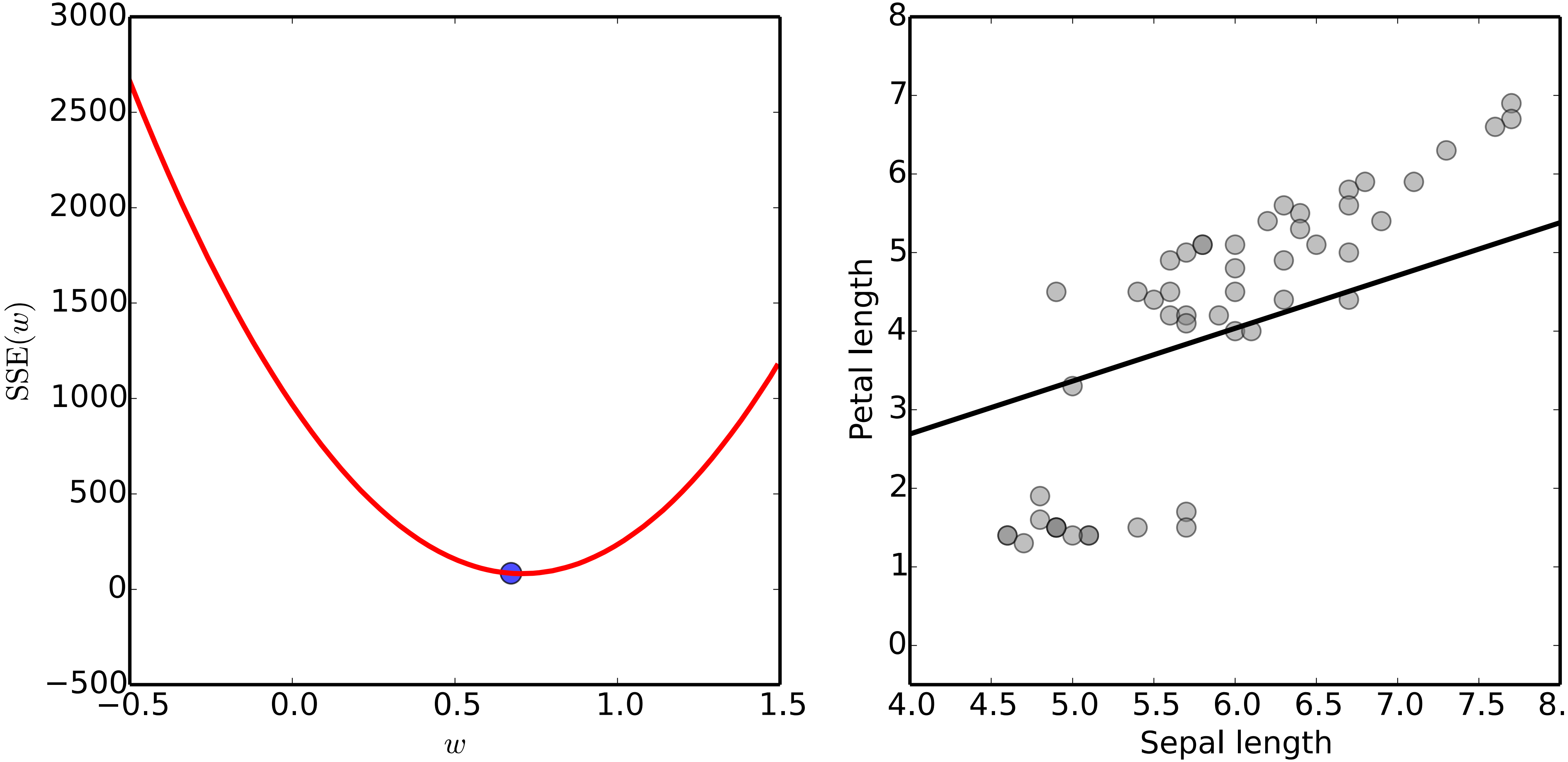
3000
2500
2000
1500
1000
500
0
−500
−0.5
0.0
0.5
1.0
1.5
$\mathrm{SSE}(w)$
w
8
7
6
5
4
3
2
1
0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Petal length
Sepal length



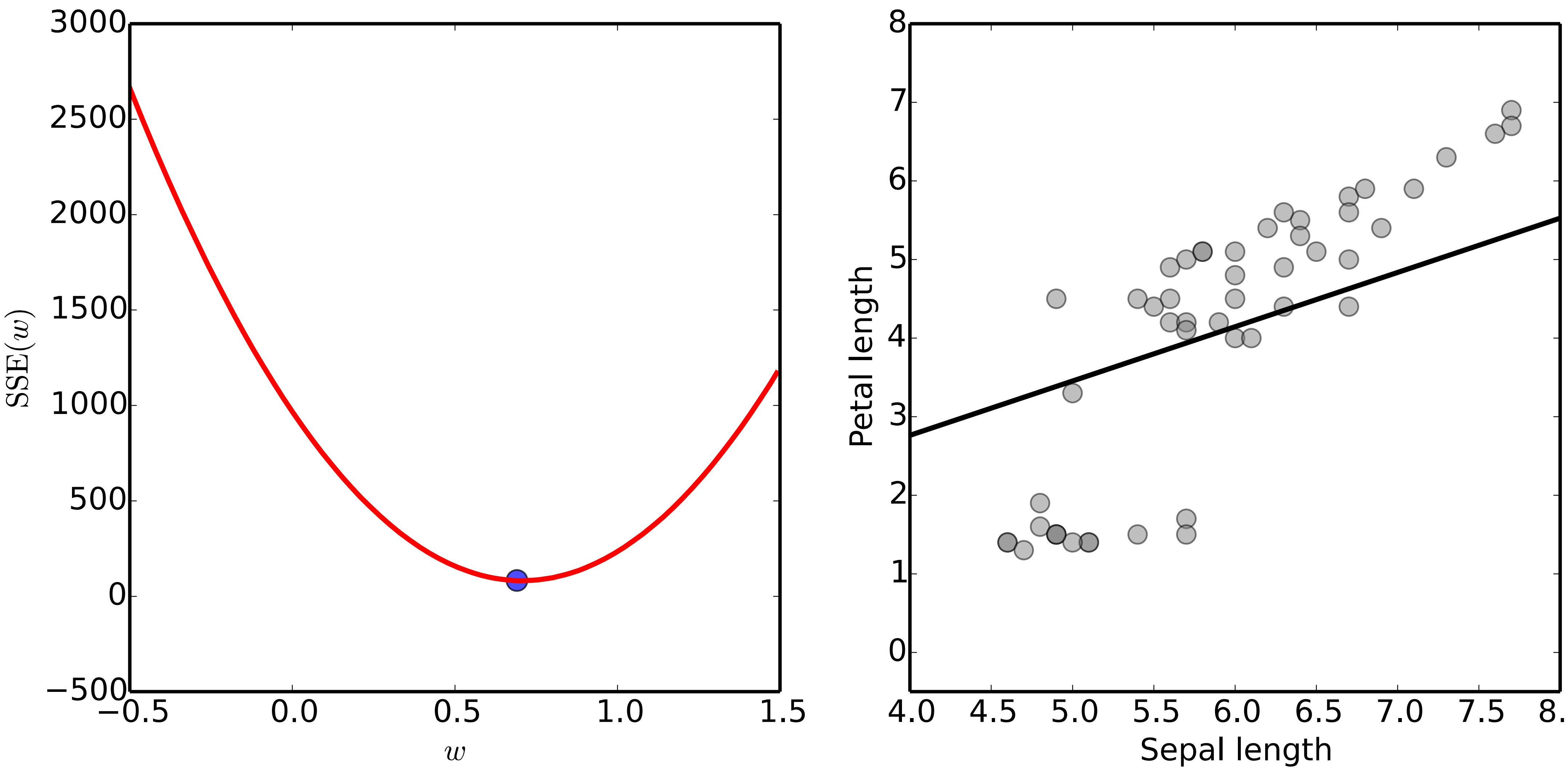
3000
2500
2000
1500
1000
500
0
−500
$\mathrm{SSE}(w)$
−0.5
0.0
0.5
1.0
1.5
w
8
7
6
5
4
3
2
1
0
Petal length
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Sepal length



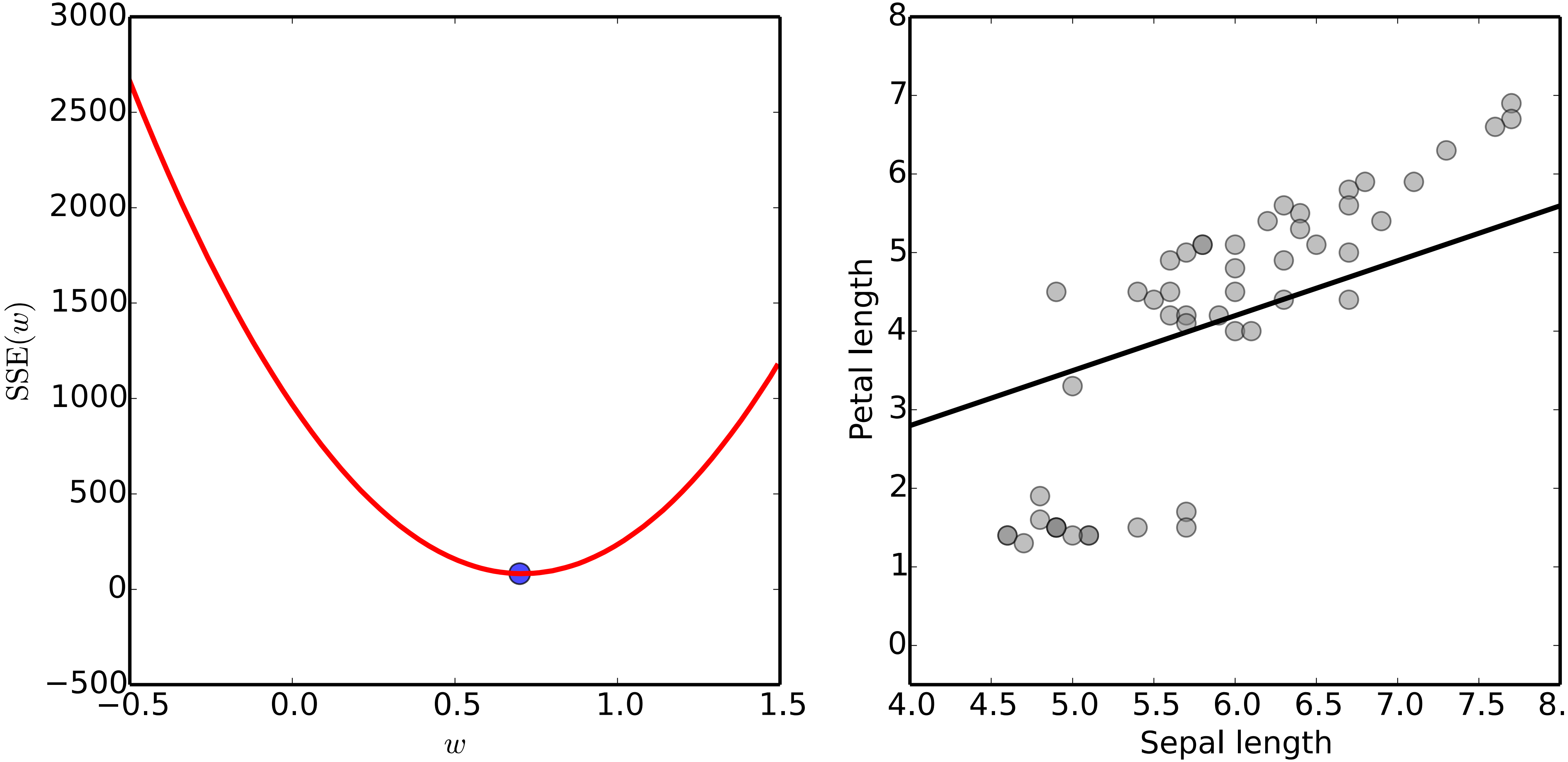
3000
2500
2000
1500
1000
500
0
−500
−0.5
0.0
0.5
1.0
1.5
w
$\mathrm{SSE}(w)$
8
7
6
5
4
3
2
1
0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Sepal length
Petal length



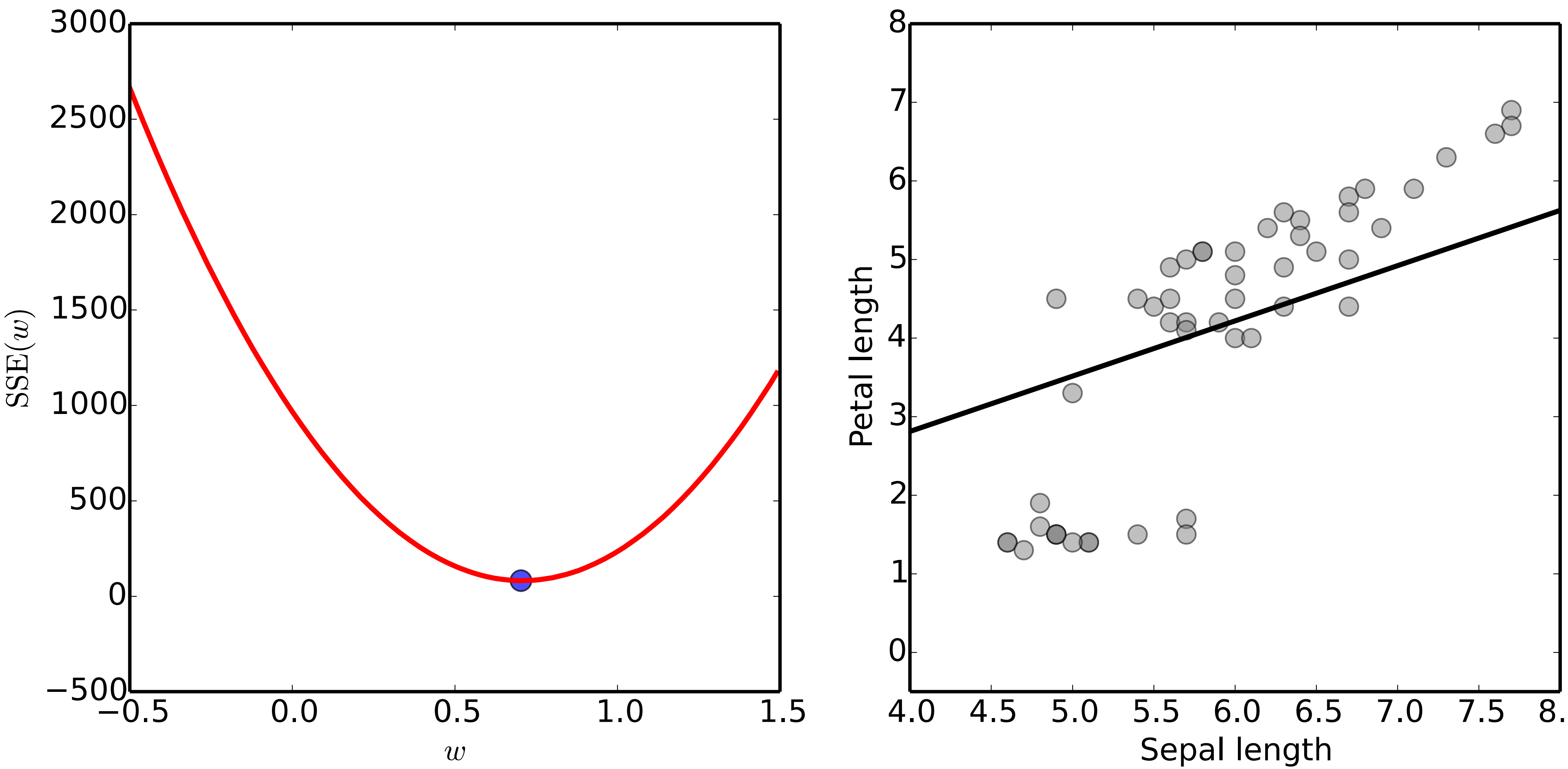
3000
2500
2000
1500
1000
500
0
−500
−0.5
0.0
0.5
1.0
1.5
$\mathrm{SSE}(w)$
w
8
7
6
5
4
3
2
1
0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Petal length
Sepal length



3000
2500
2000
1500
1000
500
0
−500
−0.5
0.0
0.5
1.0
1.5
$\mathrm{SSE}(w)$
w
8
7
6
5
4
3
2
1
0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Petal length
Sepal length



3000
2500
2000
1500
1000
500
0
−500
−0.5
0.0
0.5
1.0
1.5
$\mathrm{SSE}(w)$
w
8
7
6
5
4
3
2
1
0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Petal length
Sepal length



3000
2500
2000
1500
1000
500
0
−500
−0.5
0.0
0.5
1.0
1.5
$\mathrm{SSE}(w)$
w
8
7
6
5
4
3
2
1
0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Petal length
Sepal length

Gradient descent

- For many types of models, we can find good model parameters with gradient descent
- Important to use appropriate learning rate!

Problem: learning rate

- What will happen if the learning rate is too small?
- And if it's too big?

Gradient descent with big datasets

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \eta \times 2 \sum_{i=1}^{N} (y^i_{\text{pred}} - y^i)\mathbf{x}^i$$

We have to compute predictions and calculate residuals for all the examples before making an update.

Idea: update more often

- Instead, we could update after every example (or after every 100):

1: **for** $i = 1$ **to** N **do**

2: $\quad \mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \eta \times 2(y^i_{\text{pred}} - y^i)\mathbf{x}^i$

Stochastic Gradient Descent

- **Stochastic** gradient descent (aka **SGD**)
- Suitable for online learning scenarios
  - Compare with the Perceptron update rule
- Workhorse of modern Machine Learning
  - Large, deep neural networks

Summary

- Modular learning:
  Model + Optimization
- Gradient descent to find model parameters with lowest error
- SGD for working with big data