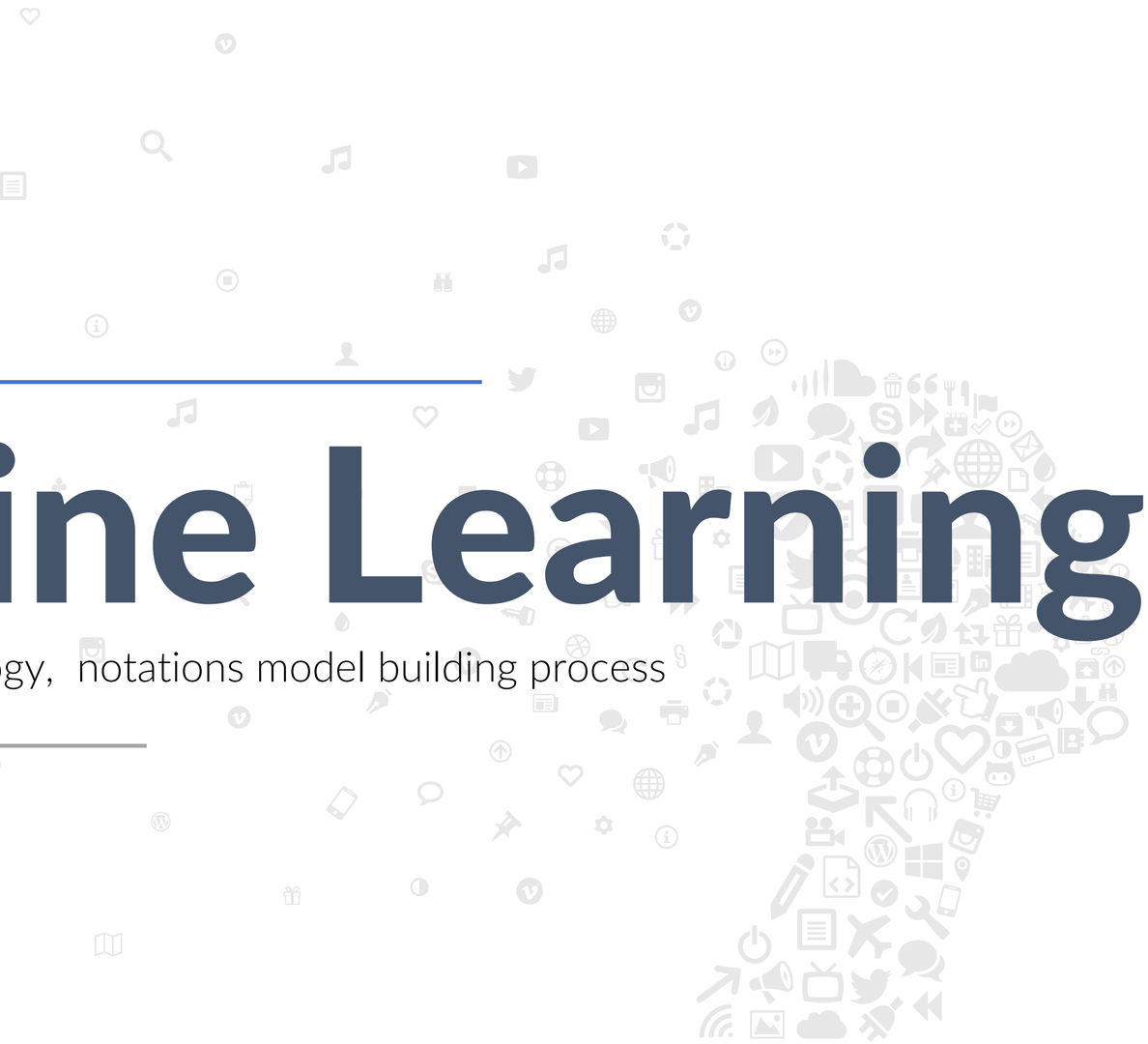
Basic Machine Learning

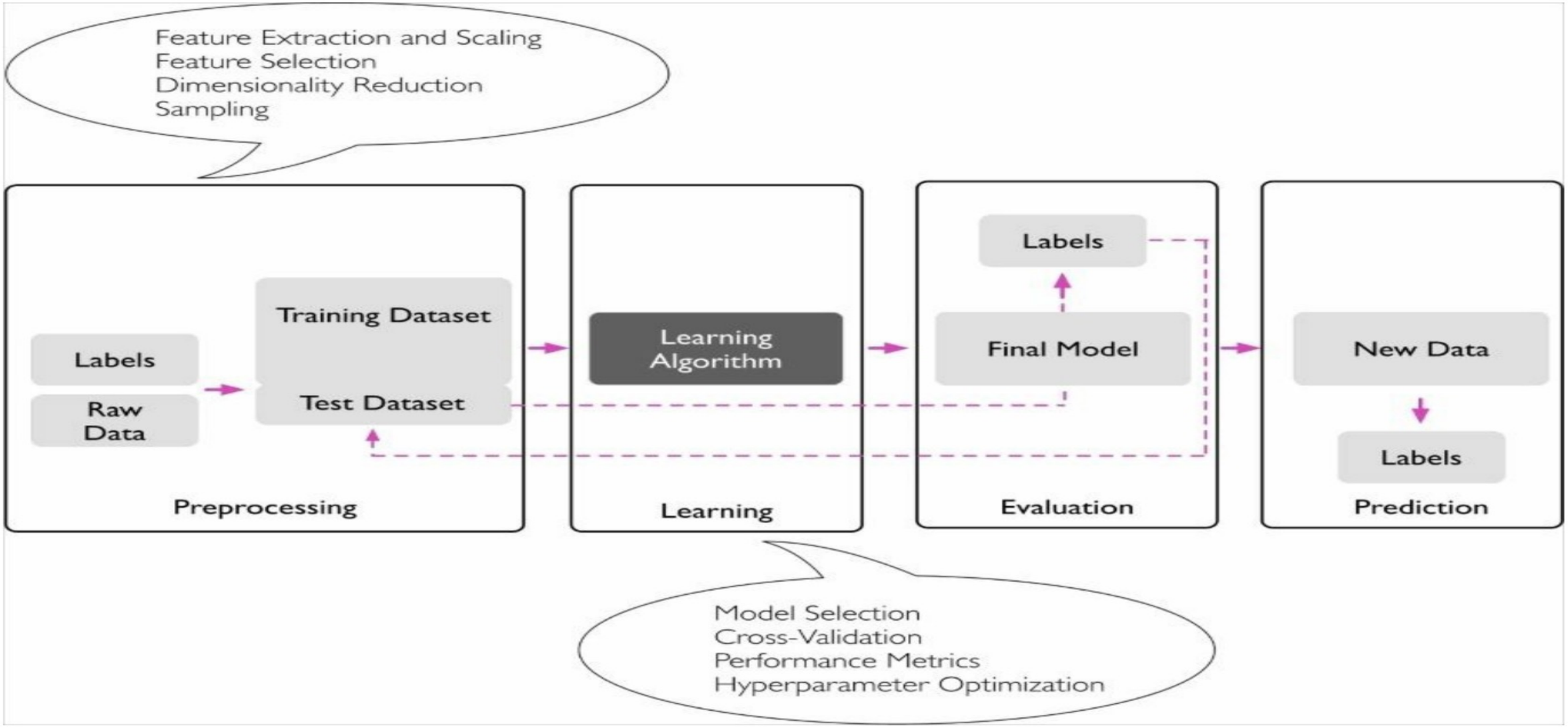
Understanding data, basic terminology, notations model building process



Kyi Thar
kyi.thar@pipe.bz

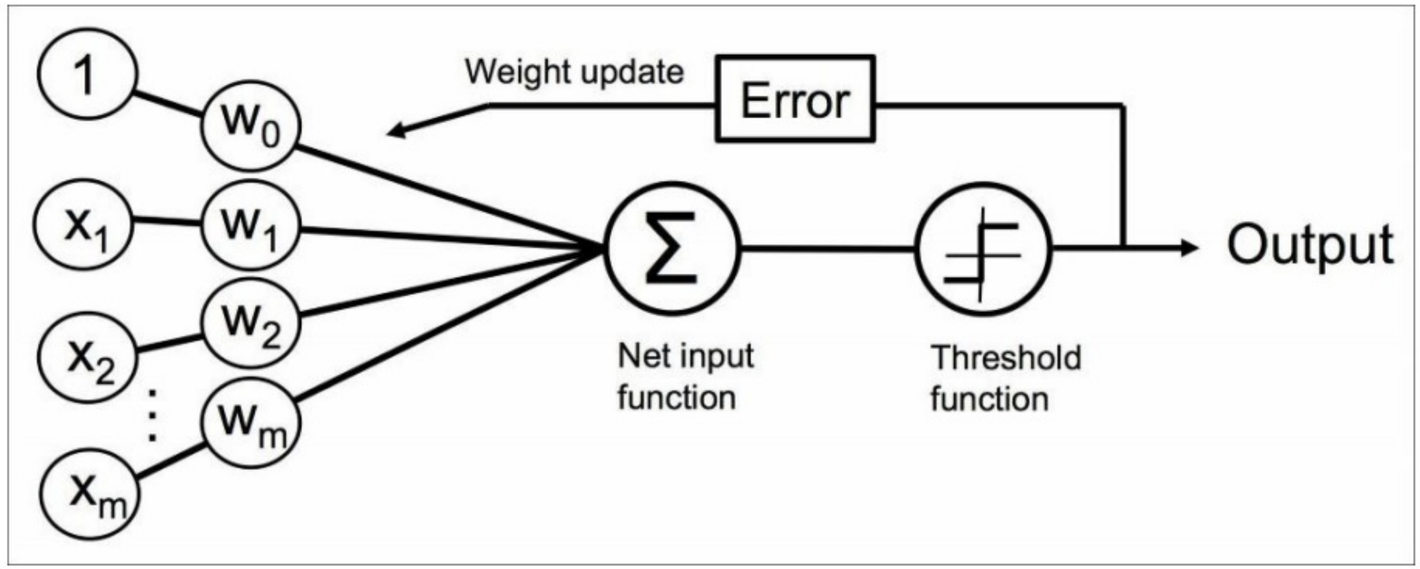
Introduction to the basic terminology and notations

Machine Learning Pipeline



Introduction to the basic terminology and notations

Model



$$\Delta w_j = \eta(-1--1)x_j^{(i)} = 0$$

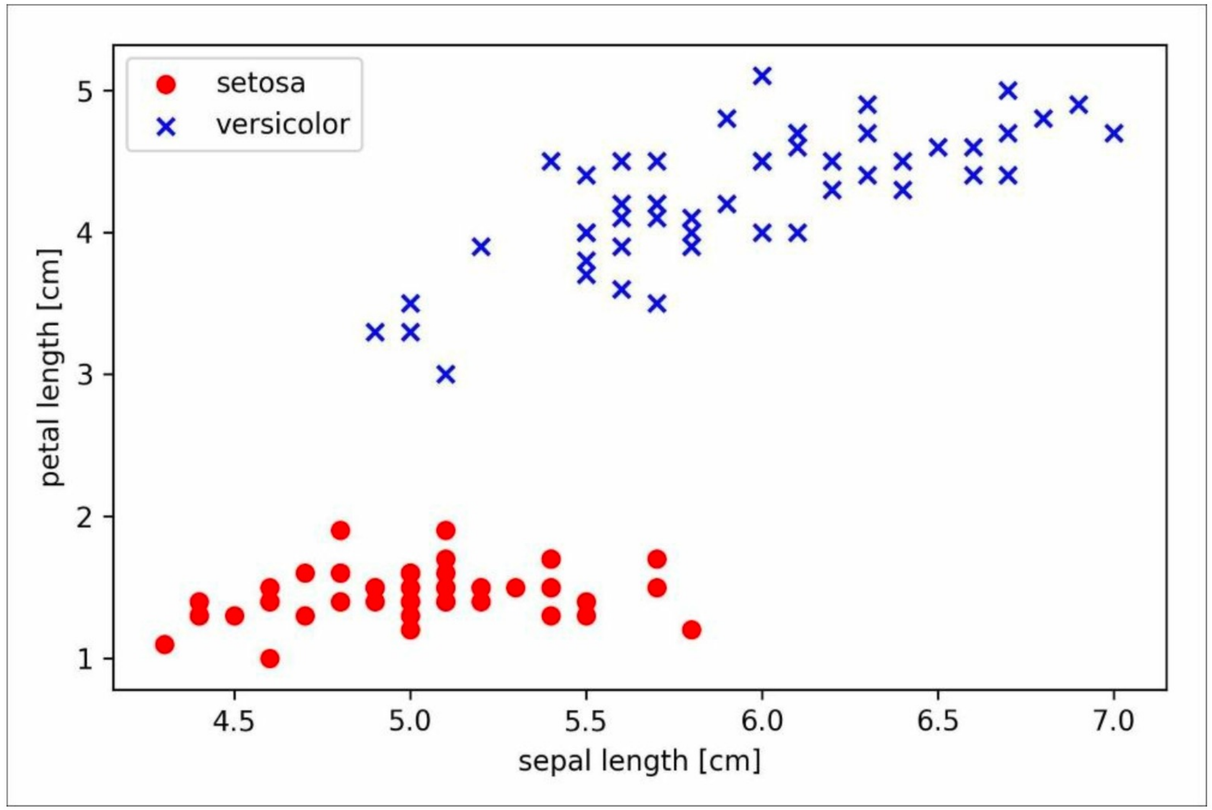
$$\Delta w_j = \eta(1-1)x_j^{(i)} = 0$$

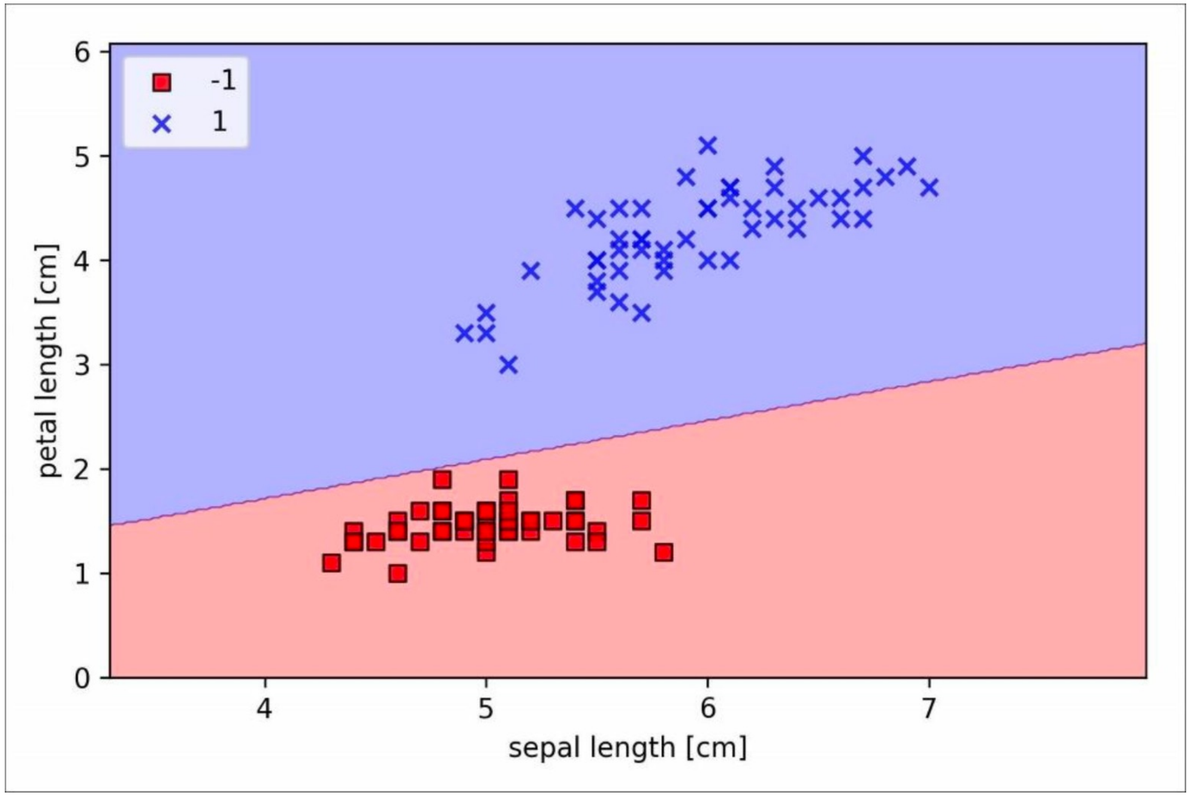
$$\Delta w_j = \eta(1--1)x_j^{(i)} = \eta(2)x_j^{(i)}$$

$$\Delta w_j = \eta(-1-1)x_j^{(i)} = \eta(-2)x_j^{(i)}$$

Introduction to the basic terminology and notations

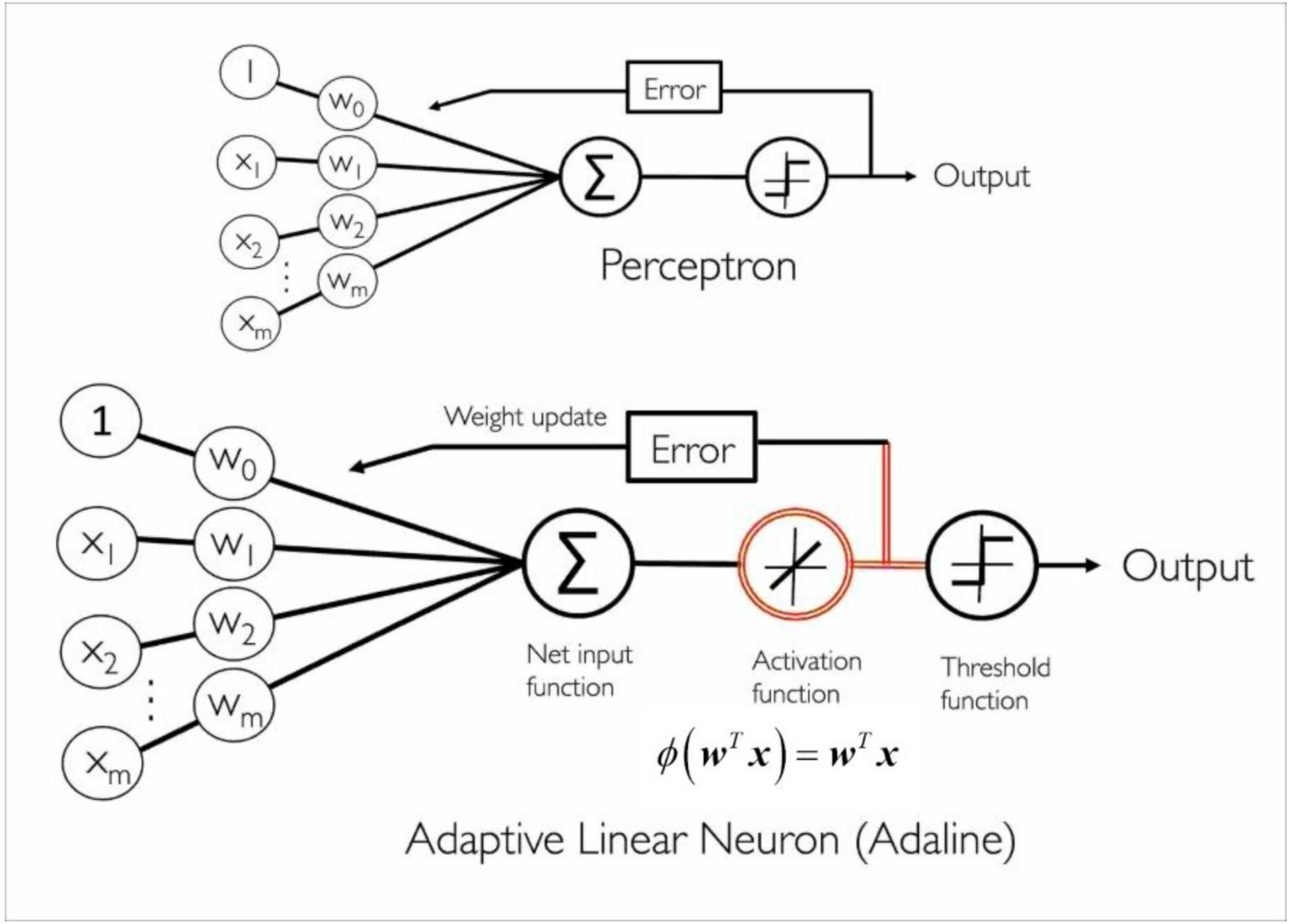
Model





Introduction to the basic terminology and notations

Adaptive Linear Neuron



Introduction to the basic terminology and notations

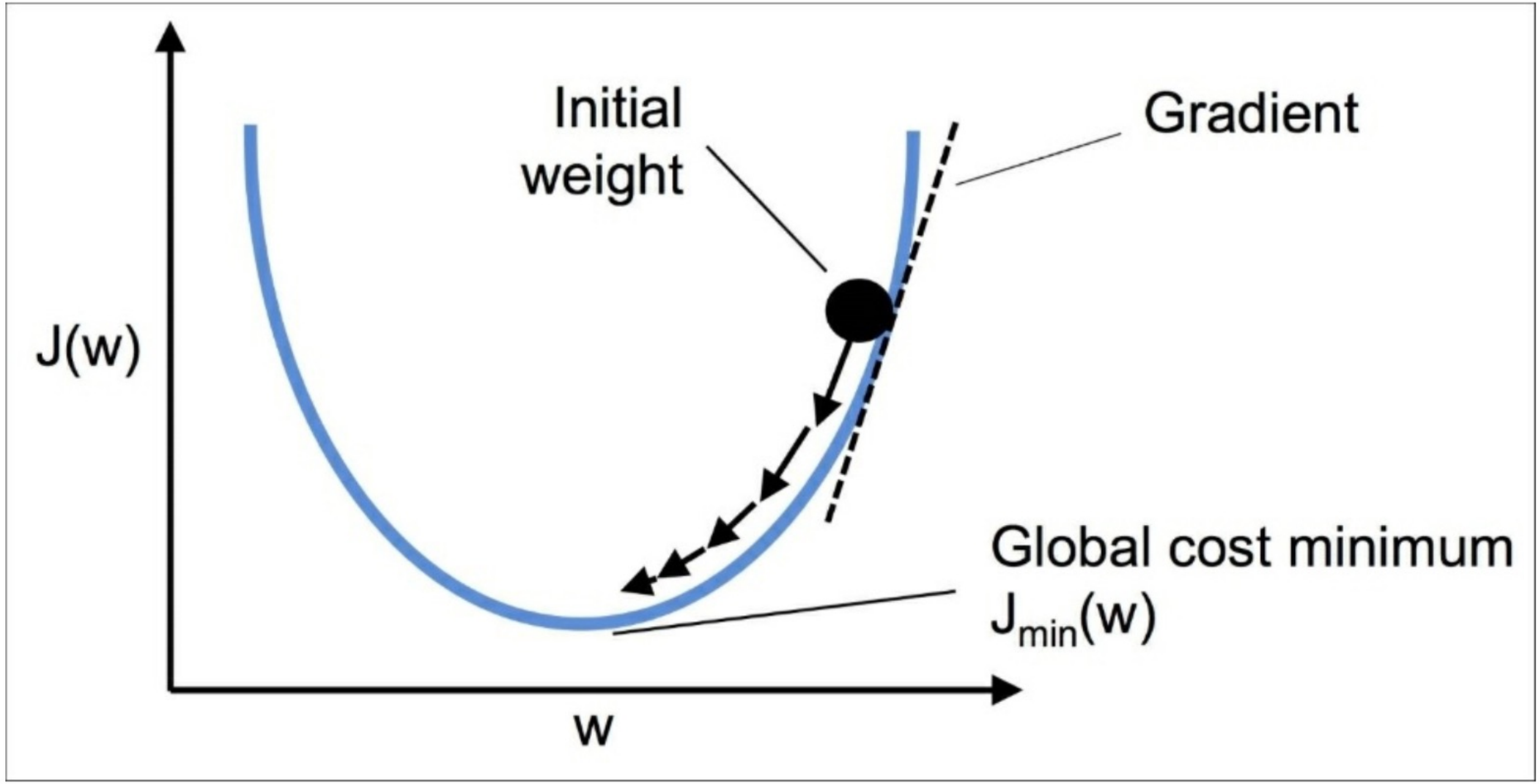
Minimizing cost functions with gradient descent

One of the key ingredients of supervised machine learning algorithms is a defined **objective function** that is to be optimized during the learning process. This objective function is often a cost function that we want to minimize. In the case of Adaline, we can define the cost function J to learn the weights as the **Sum of Squared Errors (SSE)** between the calculated outcome and the true class label:

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right)\right)^2$$

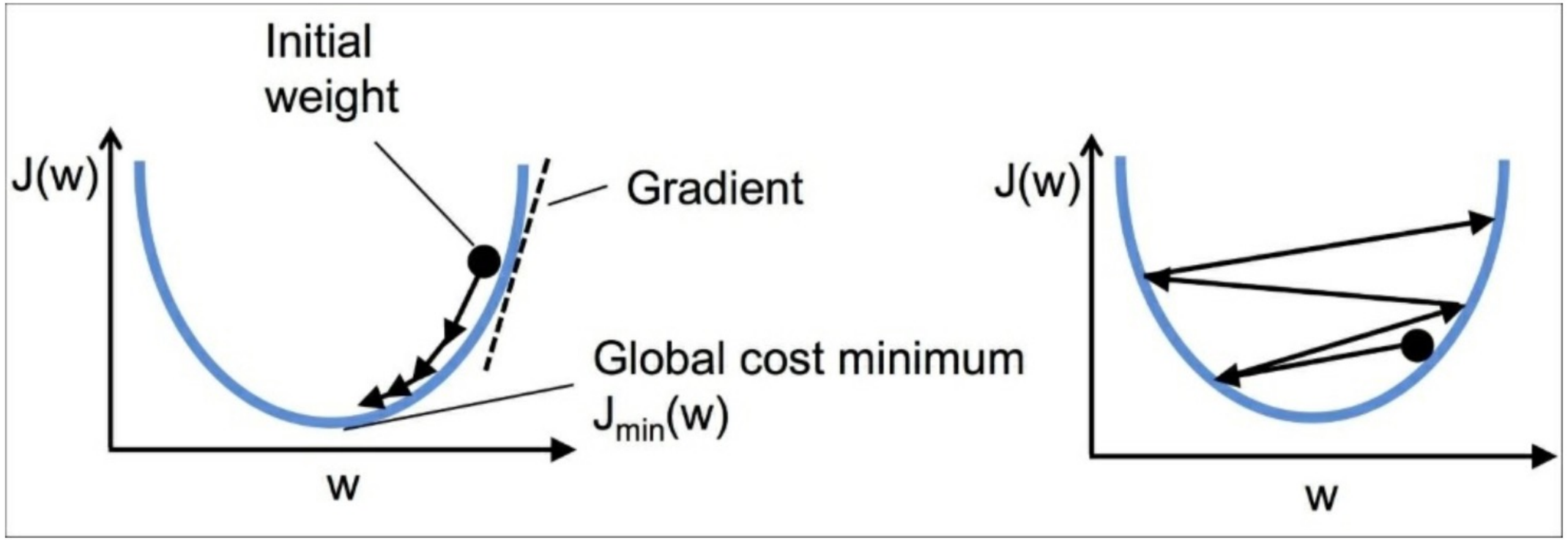
Introduction to the basic terminology and notations

Minimizing cost functions with gradient descent



Introduction to the basic terminology and notations

Learning rate



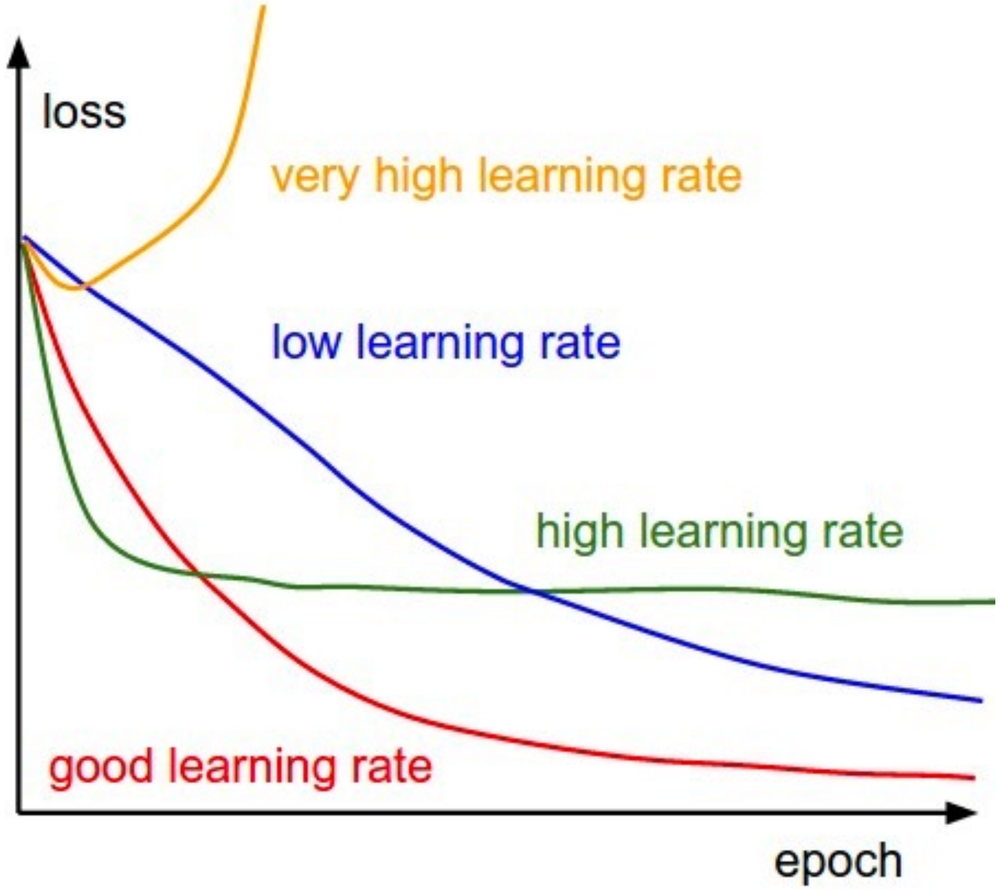
Introduction to the basic terminology and notations

Learning rate

https://the-learning-machine.com/article/optimization/gradient-descent

Introduction to the basic terminology and notations

Learning rate



https://cs231n.github.io/neural-networks-3/

Using gradient descent, we can now update the weights by taking a step in the opposite direction of the gradient $\nabla J(w)$ of our cost function $J(w)$:

$$w := w + \Delta w$$

Where the weight change Δw is defined as the negative gradient multiplied by tl learning rate η:

$$\Delta w = -\eta \nabla J(w)$$

Introduction to the basic terminology and notations

Minimizing cost functions with gradient descent

To compute the gradient of the cost function, we need to compute the partial derivative of the cost function with respect to each weight w_j:

$$\frac{\partial J}{\partial w_j} = -\sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) x_j^{(i)}$$

So that we can write the update of weight w_j as:

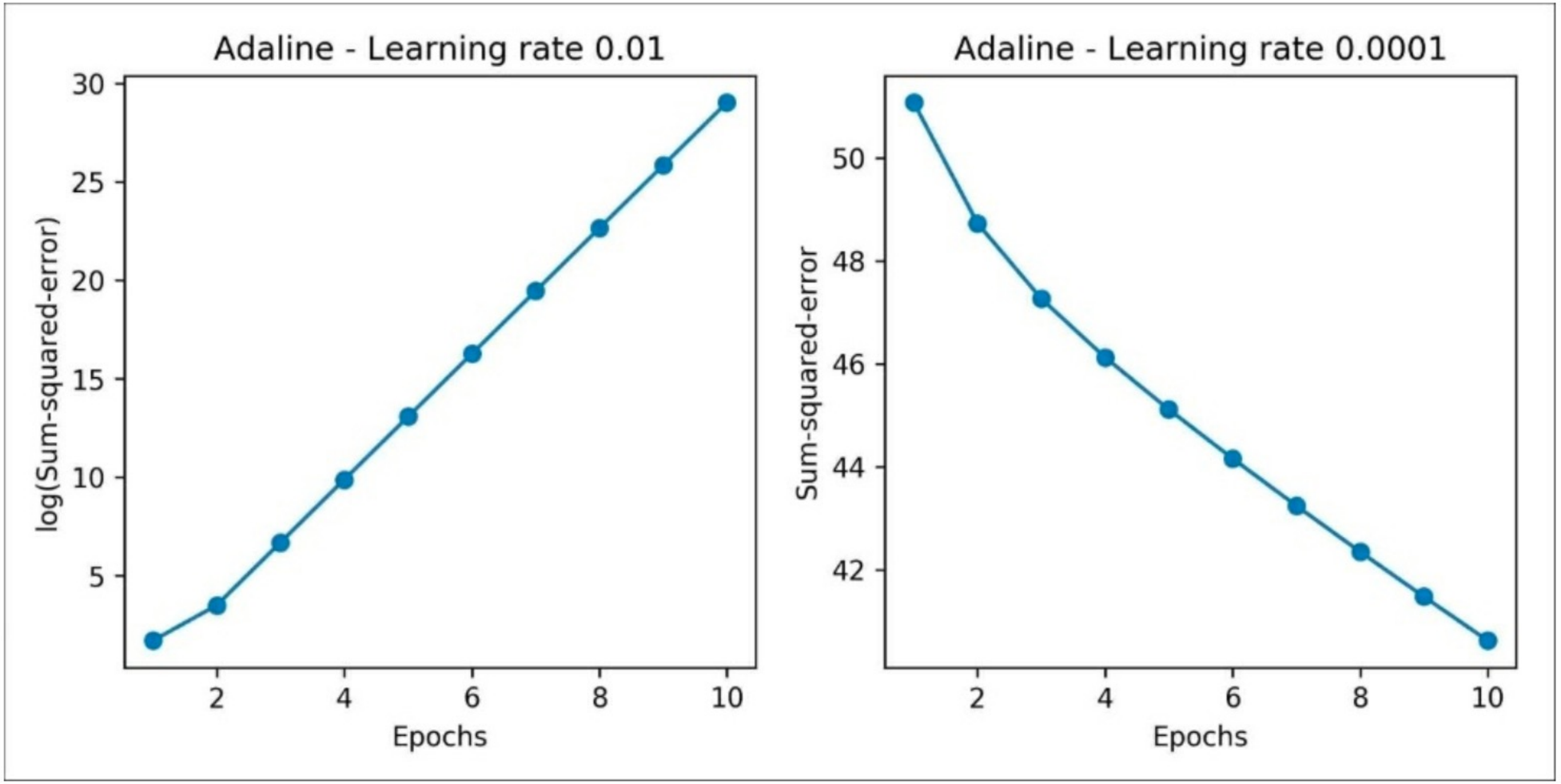
$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) x_j^{(i)}$$

Since we update all weights simultaneously, our Adaline learning rule becomes:

$$\boldsymbol{w} := \boldsymbol{w} + \Delta \boldsymbol{w}$$

Introduction to the basic terminology and notations

Minimizing cost functions with gradient descent

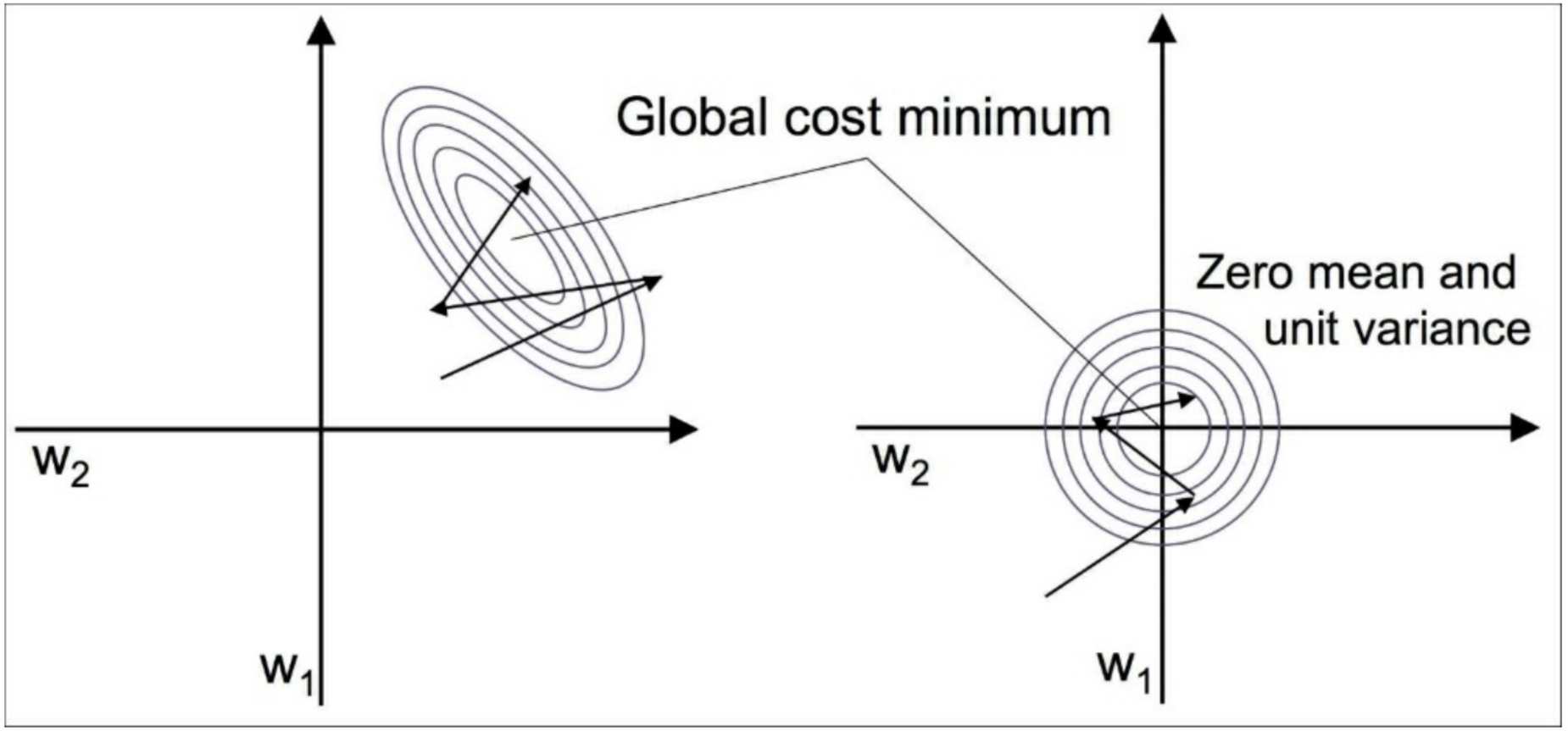


Gradient descent is one of the many algorithms that benefit from feature scaling. In this section, we will use a feature scaling method called **standardization**, which gives our data the property of a standard normal distribution, which helps gradient descent learning to converge more quickly. Standardization shifts the mean of each feature so that it is centered at zero and each feature has a standard deviation of 1. For instance, to standardize the *j*th feature, we can simply subtract the sample mean μ_j from every training sample and divide it by its standard deviation σ_j:

$$\boldsymbol{x}'_j = \frac{\boldsymbol{x}_j - \mu_j}{\sigma_j}$$

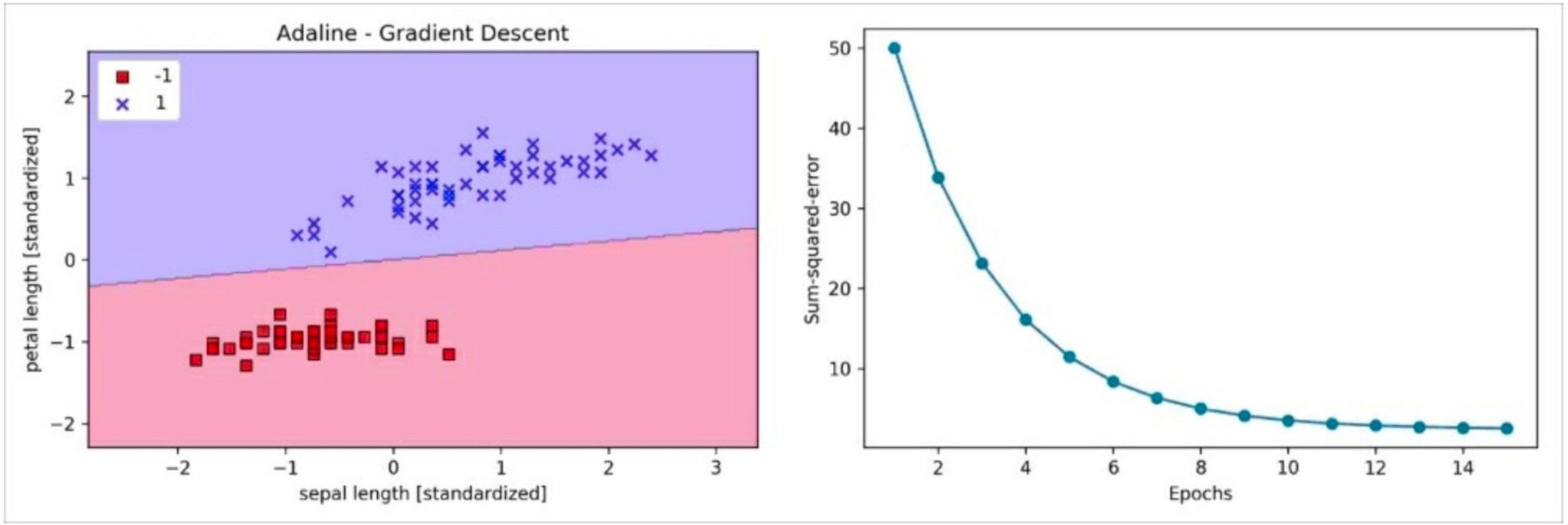
Introduction to the basic terminology and notations

Improving gradient descent through feature scaling



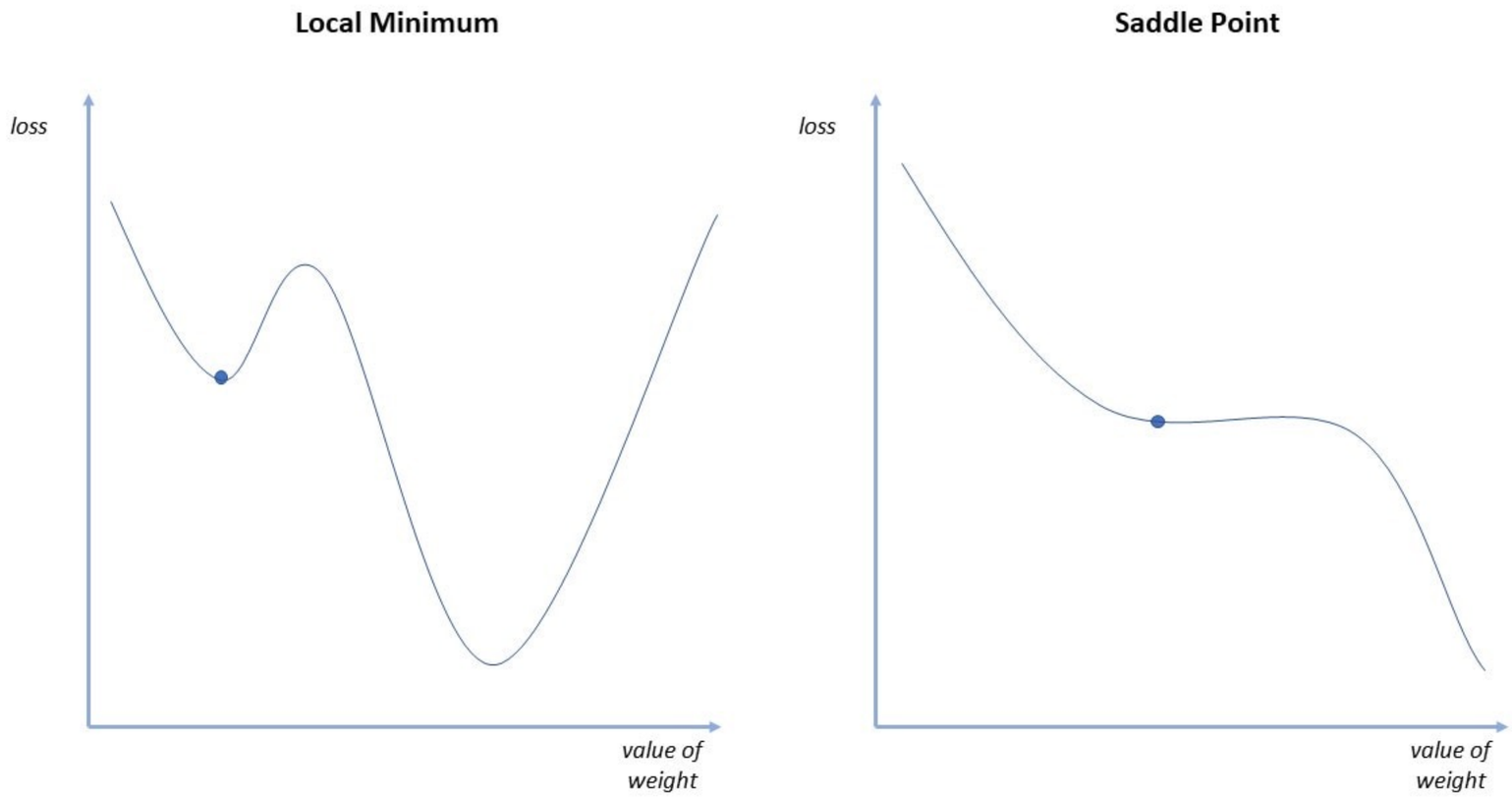
Introduction to the basic terminology and notations

Improving gradient descent through feature scaling



Introduction to the basic terminology and notations

Challenges with gradient descent



https://www.ibm.com/cloud/learn/gradient-descent

Introduction to the basic terminology and notations

In the previous section, we learned how to minimize a cost function by taking a step in the opposite direction of a cost gradient that is calculated from the whole training set; this is why this approach is sometimes also referred to as **batch gradient descent**. Now imagine we have a very large dataset with millions of data points, which is not uncommon in many machine learning applications. Running batch gradient descent can be computationally quite costly in such scenarios since we need to reevaluate the whole training dataset each time we take one step towards the global minimum.

A popular alternative to the batch gradient descent algorithm is **stochastic gradient descent**, sometimes also called iterative or online gradient descent. Instead of updating the weights based on the sum of the accumulated errors over all samples $\boldsymbol{x}^{(i)}$:

$$\Delta \boldsymbol{w} = \eta \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) \boldsymbol{x}^{(i)}$$

We update the weights incrementally for each training sample:

$$\eta\left(y^{(i)} - \phi\left(z^{(i)}\right)\right) \boldsymbol{x}^{(i)}$$

Introduction to the basic terminology and notations

In the previous section, we learned how to minimize a cost function by taking a step in the opposite direction of a cost gradient that is calculated from the whole training set; this is why this approach is sometimes also referred to as **batch gradient descent**. Now imagine we have a very large dataset with millions of data points, which is not uncommon in many machine learning applications. Running batch gradient descent can be computationally quite costly in such scenarios since we need to reevaluate the whole training dataset each time we take one step towards the global minimum.

A popular alternative to the batch gradient descent algorithm is **stochastic gradient descent**, sometimes also called iterative or online gradient descent. Instead of updating the weights based on the sum of the accumulated errors over all samples $\boldsymbol{x}^{(i)}$:

$$\Delta \boldsymbol{w} = \eta \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) \boldsymbol{x}^{(i)}$$

We update the weights incrementally for each training sample:

$$\eta\left(y^{(i)} - \phi\left(z^{(i)}\right)\right) \boldsymbol{x}^{(i)}$$

Introduction to the basic terminology and notations

In stochastic gradient descent implementations, the fixed learning rate η is often replaced by an adaptive learning rate that decreases over time, for example:

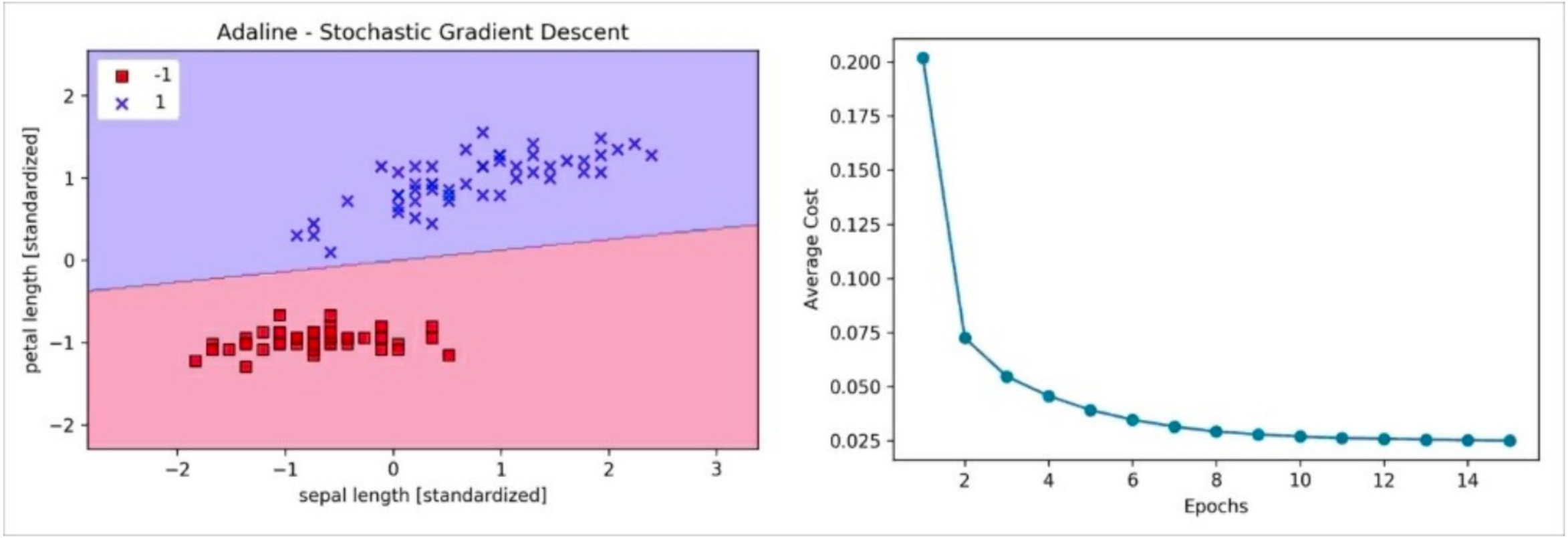
$$\frac{c_1}{[number\ of\ iterations]+c_2}$$

Where c_1 and c_2 are constants. We shall note that stochastic gradient descent does not reach the global minimum, but an area very close to it. And using an adaptive learning rate, we can achieve further annealing to the cost minimum.

Another advantage of stochastic gradient descent is that we can use it for **online learning**. In online learning, our model is trained on the fly as new training data arrives. This is especially useful if we are accumulating large amounts of data, for example, customer data in web applications. Using online learning, the system can immediately adapt to changes and the training data can be discarded after updating the model if storage space is an issue.

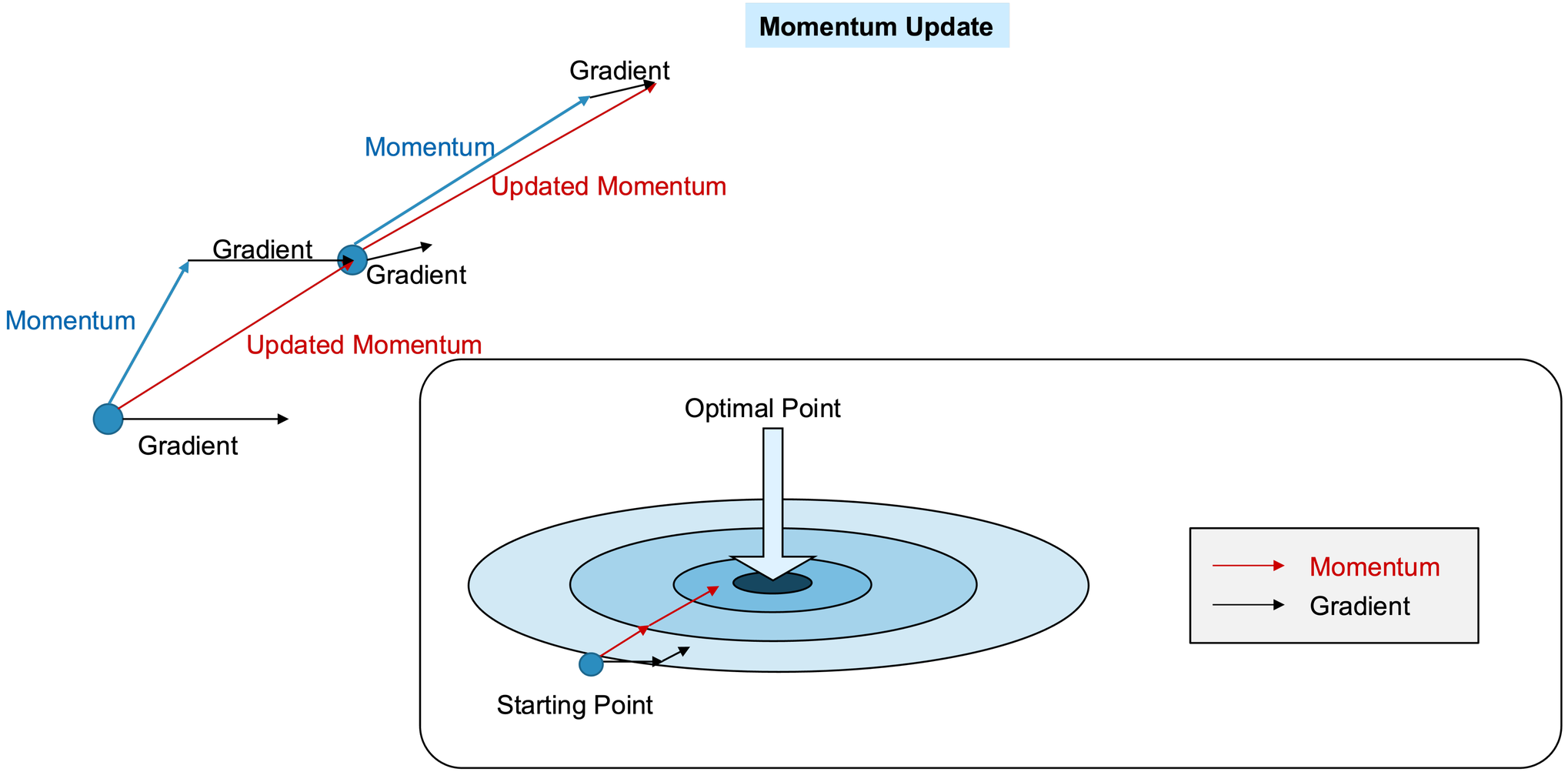
Introduction to the basic terminology and notations

Large-scale machine learning and stochastic gradient descent – adaptive learning rate



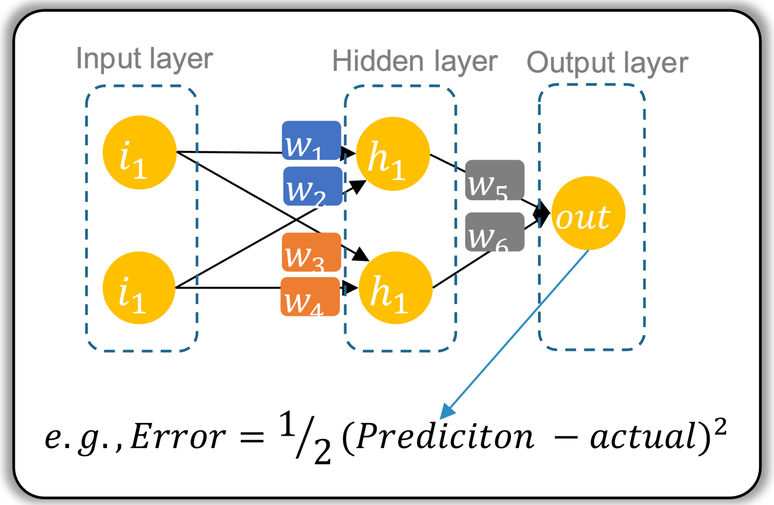
Introduction to the basic terminology and notations

SGD vs SGD with momentum



Introduction to the basic terminology and notations

SGD without momentum vs SGD with momentum



Derivative of error with respect to each weight

Old weight

$$W_x = W_x - \gamma.\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Learning rate

(a) SGD without momentum

Old momentum

Derivative of error with respect to each weight

$$v_x = v_x + \gamma.\left(\frac{\partial Error}{\partial W_x}\right)$$

New momentum

Learning rate

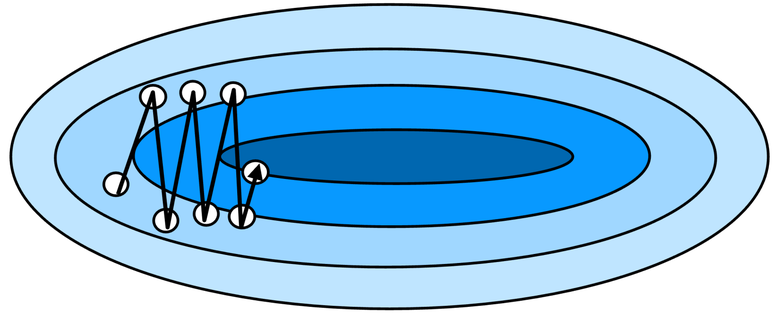
Old weight

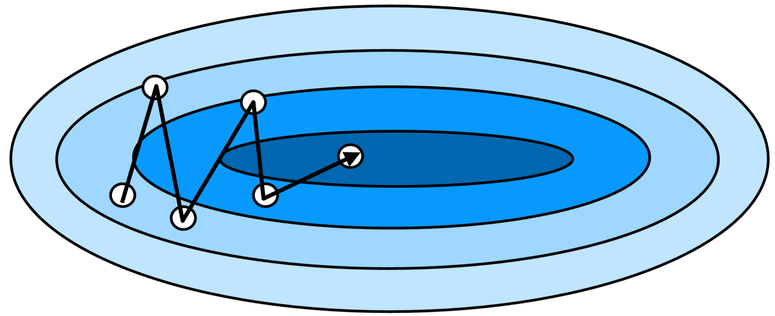
Updated momentum

$$W_x = W_x - v_x$$

New weight

(b) SGD with momentum





Source: Genevieve B. Orr

Q&A