We are starting at 13:00!

Grab a seat and get ready

Saturdays.AI
Kigali

Saturdays.AI
Kigali

#2 Simple ML for Classification

AI Saturdays Kigali

Simple ML for Classification (Linear Classifiers)

Saturdays.AI
Kigali

Linear Classifiers in Python

In today's class, we'll explore two foundational classification algorithms in machine learning:

- **Perceptron**
- **Adaptive Linear Neuron (Adaline)**

What You'll Learn

- Step-by-step **implementation of the Perceptron** in Python
    - Apply it to classify flower species using the Iris dataset
    - Understand how basic ML algorithms work and how to code them
- Introduction to **optimization** with Adaline
    - Prepares you for using **advanced models** with **scikit-learn**

Tools & Topics Covered

- Building an understanding of machine learning algorithms
- Using **Pandas**, **NumPy**, and **Matplotlib** to read in, process, and visualize data
- Implementing **linear classifiers** for 2-class problems in Python

Saturdays.AI
Kigali

A Brief History of Artificial Neurons

- **1943**: McCulloch & Pitts introduced the **first artificial neuron model**, inspired by how biological neurons work
    - Their neuron acted like a **logic gate**: combining inputs and firing if a threshold was reached

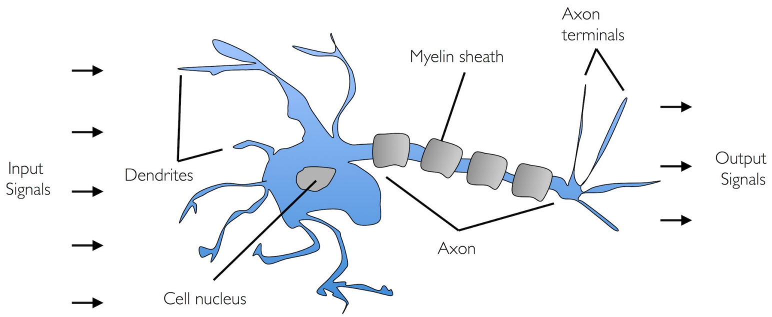


Figure 2.1: A neuron processing chemical and electrical signals

- **1957**: Frank Rosenblatt developed the **Perceptron**
    - Built on the McCulloch-Pitts (MCP) neuron
    - Introduced a learning rule to **adjust weights automatically**
    - Could be used for **binary classification** in supervised learning

These early ideas laid the foundation for modern neural networks and classification algorithms.

Saturdays.AI Kigali

Formal Definition of an Artificial Neuron

We model an artificial neuron for binary classification (Class 0 or 1) using:

Net Input (Linear Combination)

$$z = w_1x_1 + w_2x_2 + \dots + w_mx_m:$$

Where:

- x = the input features
- w =the learned weights

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Decision Function, $\sigma(\cdot)$

- Predict class 1 if $z \geq \theta$, otherwise class 0
- This is a **unit step function**:

$$\sigma(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

Saturdays.AI
Kigali

Formal Definition of an Artificial Neuron

Simplifying for Implementation

1. Move the threshold to the left:

$$z \geq \theta$$
$$z - \theta \geq 0$$

2. Introduce a bias term $b = -\theta$

$$z = w_1 x_1 + \ldots + w_m x_m + b = \mathbf{w}^T \mathbf{x} + b$$

3. Final decision function:

$$\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

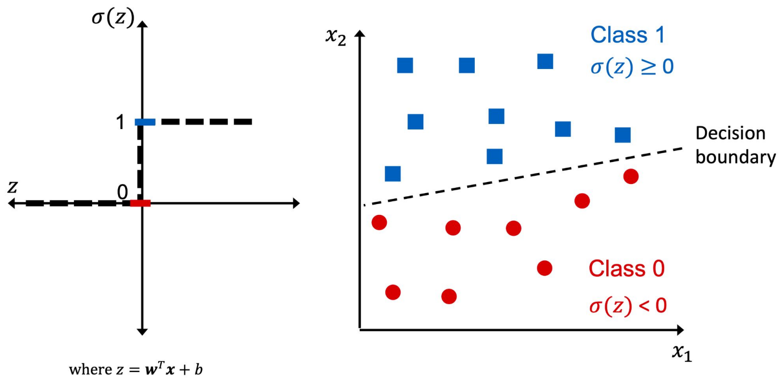


Figure 2.2: A threshold function producing a linear decision boundary for a binary classification problem

Saturdays.AI Kigali

The Perception Learning Rule

Saturdays.AI
Kigali

The Perceptron Learning Rule

Inspired by how biological neurons fire or don't, the perceptron uses a simple threshold-based model to perform binary classification.

Algorithm Overview

1. Initialize weights and bias to **zeros** (0) or **small random values**
2. For **each training example,** $\boldsymbol{x}^{(i)}$:
    a. Compute the output value, $\hat{y}^{(i)}$
    b. Update the weights and bias unit

The Perceptron Learning Rule

Update Equations

- **Weight and Bias update:**

$$w_j := w_j + \Delta w_j$$
$$\text{and} \quad b := b + \Delta b$$

- **The update values ("deltas"):**

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$$
$$\text{and} \quad \Delta b = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)$$

- η = **learning rate** (typically a constant between 0.01 and 1.0)
- $y^{(i)}$ = **true label**
- $\hat{y}^{(i)}$ = **predicted label**

Saturdays.AI
Kigali

The Perceptron Learning Rule

Convergence Condition

- The perceptron **only converges** if the two classes are **linearly separable**
- If not:
    - Set a **maximum number of epochs**, or
    - Use a **a threshold** for the number of tolerated misclassifications

      → Otherwise, the perceptron **would never stop updating the weights**

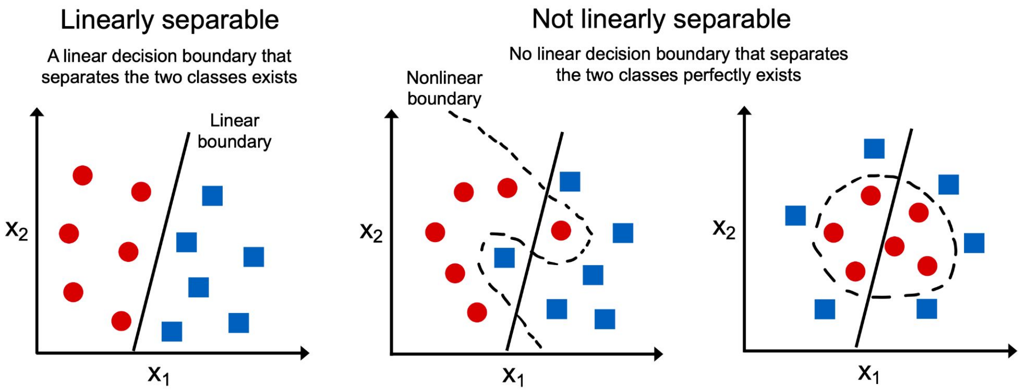


Figure 2.3: Examples of linearly and nonlinearly separable classes

Saturdays.AI Kigali

The Perceptron Learning Rule

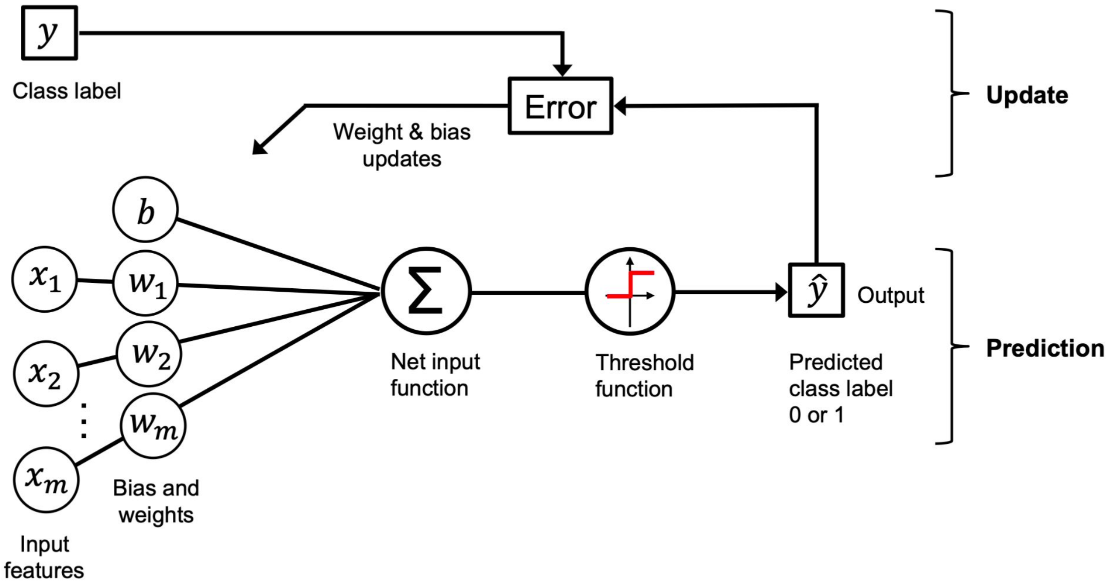


Figure 2.3: Examples of linearly and nonlinearly separable classes

Saturdays.AI Kigali

Practice

Training a perceptron model on the Iris dataset

Implementing the Perceptron algorithm in Python.

[Perceptron Algorithm]

Saturdays.AI
Kigali

Adaptive linear neuron (Adaline)

Saturdays.AI
Kigali

Why Adaline Matters

- Introduces the concept of a **continuous loss function**
- Forms the basis for understanding:
    - **Logistic regression**
    - **Support Vector Machines**
    - **Multilayer neural networks**
    - **Linear regression**

Saturdays.AI
Kigali

How Adaline Learns (vs. Perceptron)

- **Adaline**:
  - Uses a **linear activation function** for weight updates
  - Compares the **true label** to the **continuous output** of the activation function
  - Computes the error and updates weights accordingly
  - Final prediction still uses a **threshold function** (like the perceptron)

- **Perceptron**:
  - Uses a **step function** to update weights
  - Compares the **true label** to the **binary predicted label** (0 or 1)
  - Only updates if prediction is wrong

Saturdays.AI
Kigali

How Adaline Learns (vs. Perceptron)

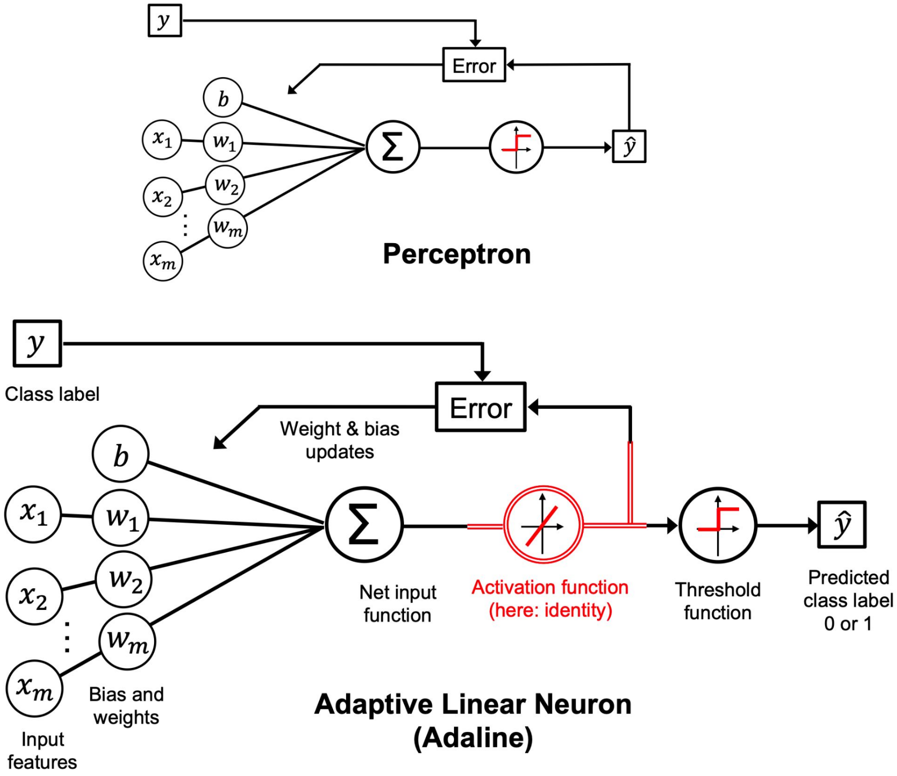


Figure 2.4: A comparison between a perceptron and the Adaline algorithm

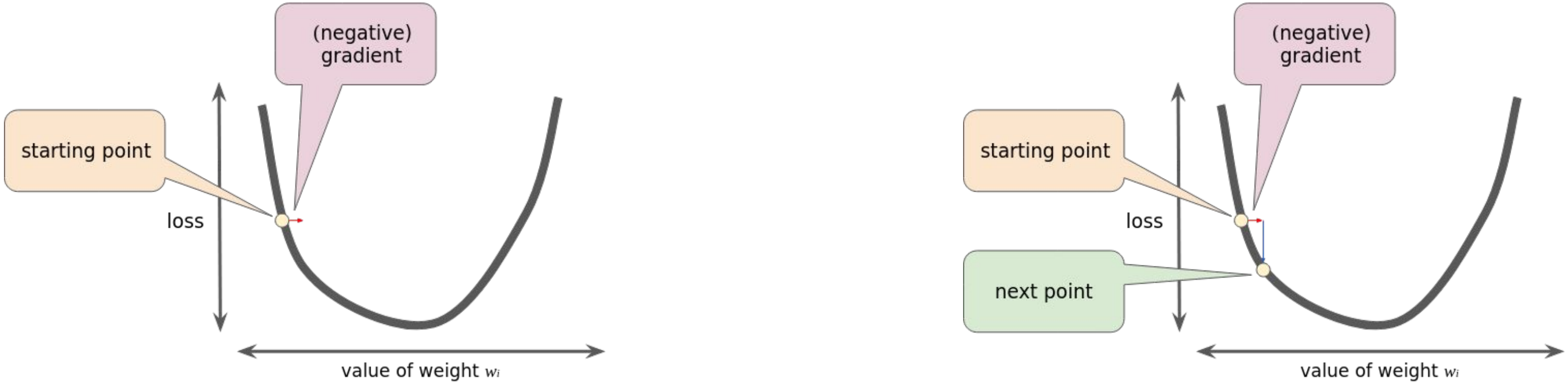
Saturdays.AI
Kigali

Gradient Descent

Saturdays.AI
Kigali

Gradient Descent: Core Idea

- Think of it as **descending a hill** until a **local or global loss minimum** is reached.
- At each step, we move in the **opposite direction of the gradient** $\nabla L(\mathbf{w}, b)$
- The size of the step depends on:
  - The **learning rate** η
  - The **slope** of the loss surface (gradient)



Saturdays.AI Kigali

Gradient Descent: Update Rules

To minimize $L(\mathbf{w}, b)$: :

$$\frac{\partial L}{\partial w_j} = -\frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right) x_j^{(i)}$$

$$\frac{\partial L}{\partial b} = -\frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)$$

$$\Delta\mathbf{w} = -\eta\nabla_w L(\mathbf{w}, b), \quad \Delta b = -\eta\nabla_b L(\mathbf{w}, b)$$

$$\Delta w_j = -\eta\frac{\partial L}{\partial w_j} \text{ and } \Delta b = -\eta\frac{\partial L}{\partial b}$$

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

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, \quad b := b + \Delta b$$

Learning Rate

Gradient descent algorithms multiply the gradient by a scalar known as the **learning rate** (also sometimes called **step size**) to determine the next point.

- For example, if the gradient magnitude is 2.5 and the learning rate is 0.01, then the gradient descent algorithm will pick the next point 0.025 away from the previous point.

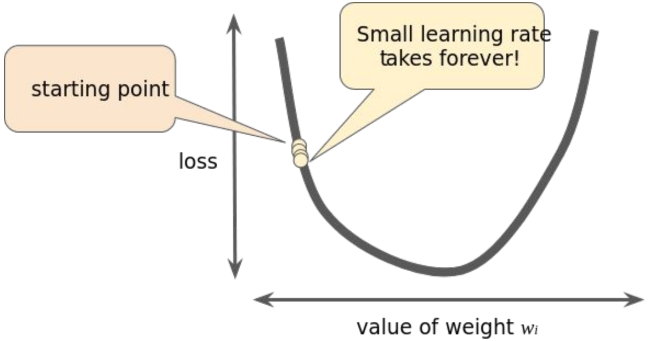
Hyperparameters are the knobs that programmers tweak in machine learning algorithms.

- Three common hyperparameters are:
  - **Learning rate**
  - **Batch size**
  - **Epochs**
- Most machine learning programmers spend a fair amount of time tuning the learning rate. If you pick a learning rate that is too small, learning will take too long:

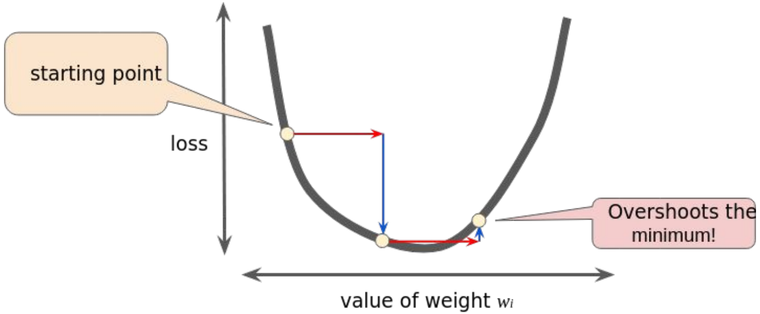
Saturdays.AI Kigali

Learning Rate

If you pick a learning rate that is too small, learning will take too long:



Conversely, if you specify a learning rate that is too large, the next point will perpetually bounce haphazardly across the bottom of the well



Saturdays.AI
Kigali

Learning Rate

An **ideal** learning rate

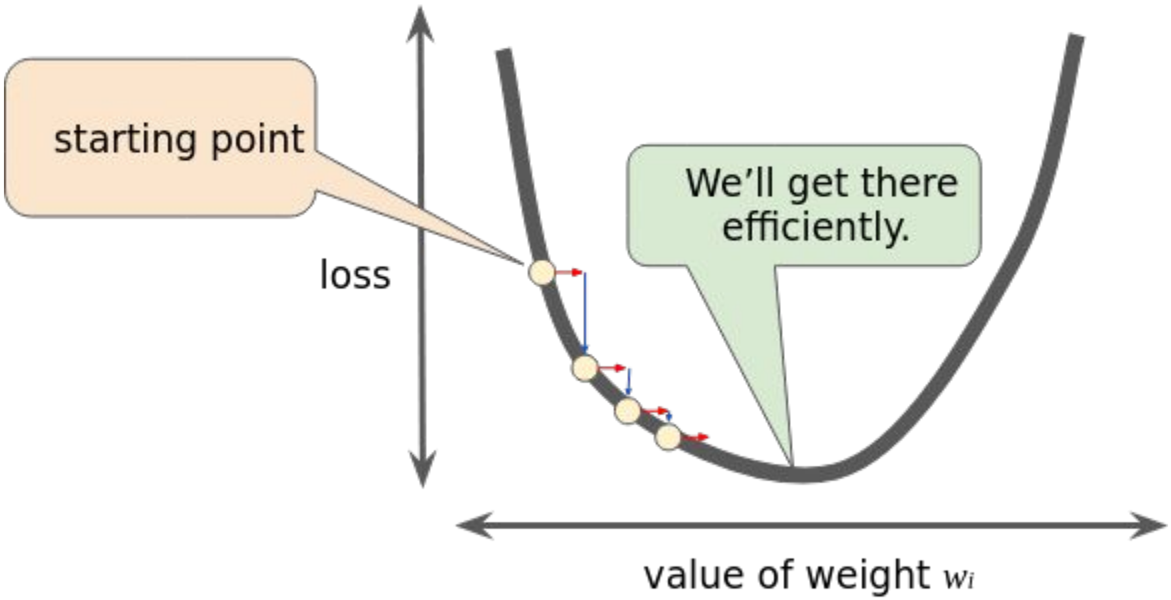
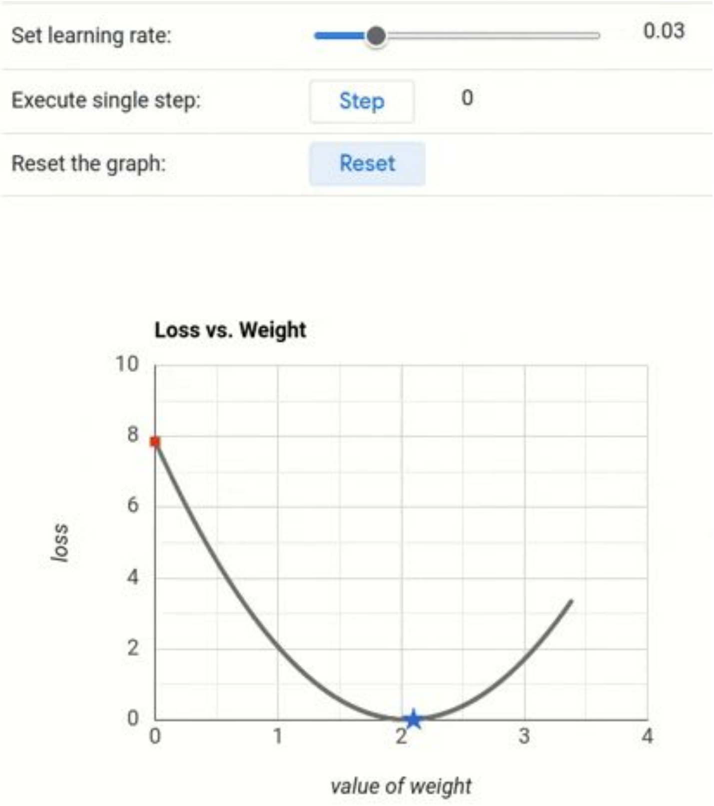


Figure 8: Learning rate is just right.

Optimizing Learning Rate

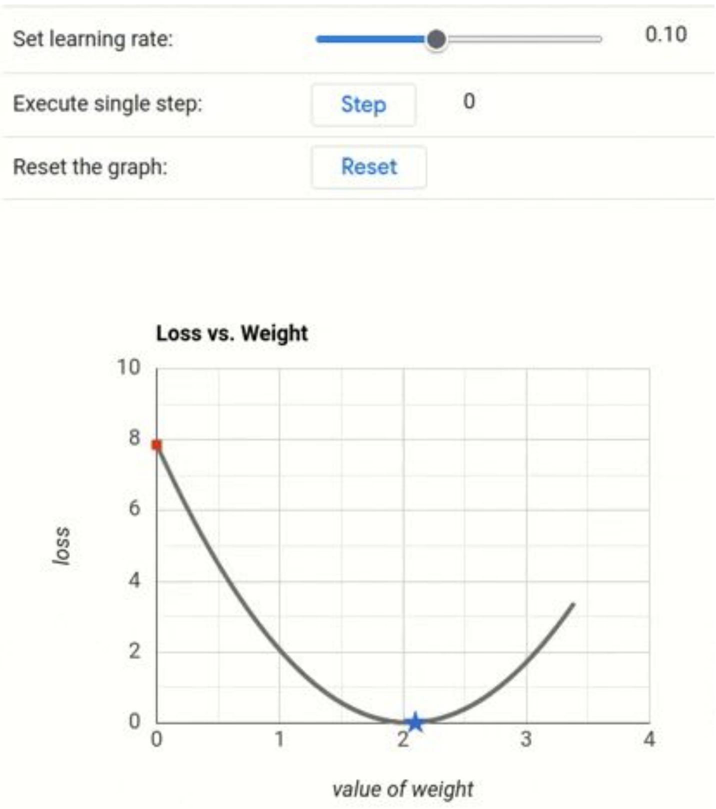
lr = 0.03



Saturdays.AI
Kigali

Optimizing Learning Rate

lr = 0.1



Saturdays.AI
Kigali

Interactive Practice

How quickly does the model converges?

The learning rate hyperparameter.

[Learning Rate]

Improving Gradient Descent: Feature Scaling

🚀 Why Feature Scaling Matters

- Many ML algorithms, including **gradient descent**, perform better with **scaled features**
- Without scaling, features on different ranges can:
    - Slow convergence
    - Cause unstable or inefficient updates

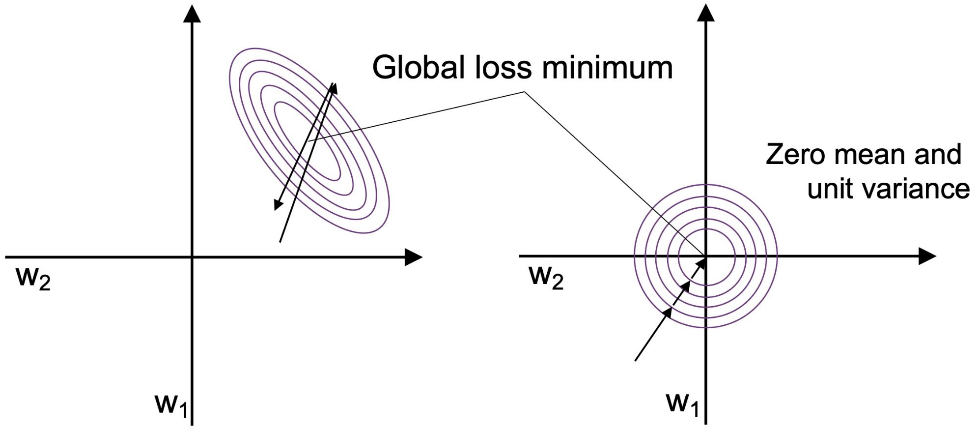


Figure 2.4: A comparison between a perceptron and the Adaline algorithm

Saturdays.AI
Kigali

Standardization: A Common Approach

- Shifts each feature to have:
  - **Mean = 0**
  - **Standard deviation = 1**
- Formula for the j-th feature:

$$x_j' = \frac{x_j - \mu_j}{\sigma_j}$$

where μ_j: is the mean and σ_j: is the standard deviation

📌 Note: Standardization **does not** make the data normally distributed — it just centers and scales it.

How It Helps Gradient Descent

- Makes it easier to choose a **single learning rate** that works across all weights
- Prevents some weights from updating too slowly or too aggressively
- Leads to **faster, more stable convergence** toward the optimal solution

Saturdays.AI
Kigali

Large-Scale ML

In gradient descent, a **batch** is the total number of examples you use to calculate the gradient in a single iteration.

Full Batch Gradient Descent (Recap)

- Involves computing the **gradient over the entire training set**
- Each step toward the minimum uses **all data points**
- Also called batch **gradient descent**

The Challenge with Large Datasets

- Modern ML often involves **millions of examples**
- Running full-batch gradient descent:
  - Is **computationally expensive**
  - Requires a full pass through the dataset **for every update step**

Saturdays.AI
Kigali

Gradient Descent Types

In gradient descent, a **batch** is the total number of examples you use to calculate the gradient in a single iteration.

- **Batch Gradient Descent**: Batch Size = Size of Training Set
- **Stochastic Gradient Descent (SDG)**: Batch Size = 1
- **Mini-Batch Gradient Descent**: 1 < Batch Size < Size of Training Set

Why Use SGD?

- **Faster convergence** due to more frequent updates
- Can **escape shallow local minima** in non-convex loss surfaces (thanks to noisy steps)

Practical Considerations

- **Shuffle training data** before each epoch to avoid cycles
- Present examples in **random order** for better generalization

Practice

Minimizing loss functions with gradient descent

Implementing Adaline in Python.

[Adaline]

Saturdays.AI
Kigali

Break

Saturdays.AI
Kigali

Practice

We give you code. What happens?

Run. Learn. Repeat.

Challenges & Next steps!

Saturdays.AI
Kigali

Best part of the day! 🥳

Kahoot!

kahoot.it

Saturdays.AI
Kigali

Any questions?

THANKS

Saturdays.AI
Kigali

kigali@saturdays.ai
coming soon
coming soon