UNIVERSITY of WEST FLORIDA

Training Simple ML Algorithm for Classification

Module 2

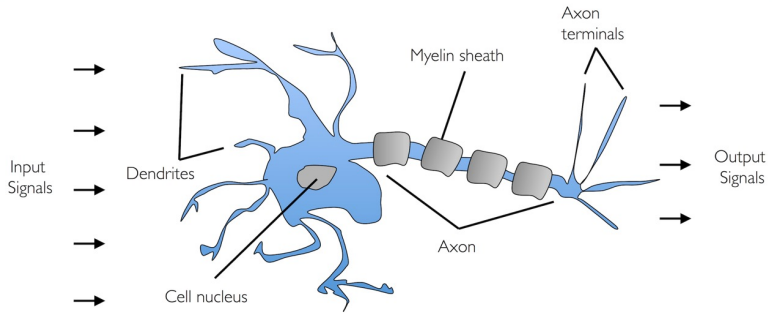
Brian Jalaian, Ph.D.
Associate Professor
Intelligent Systems & Robotics Department

Outline

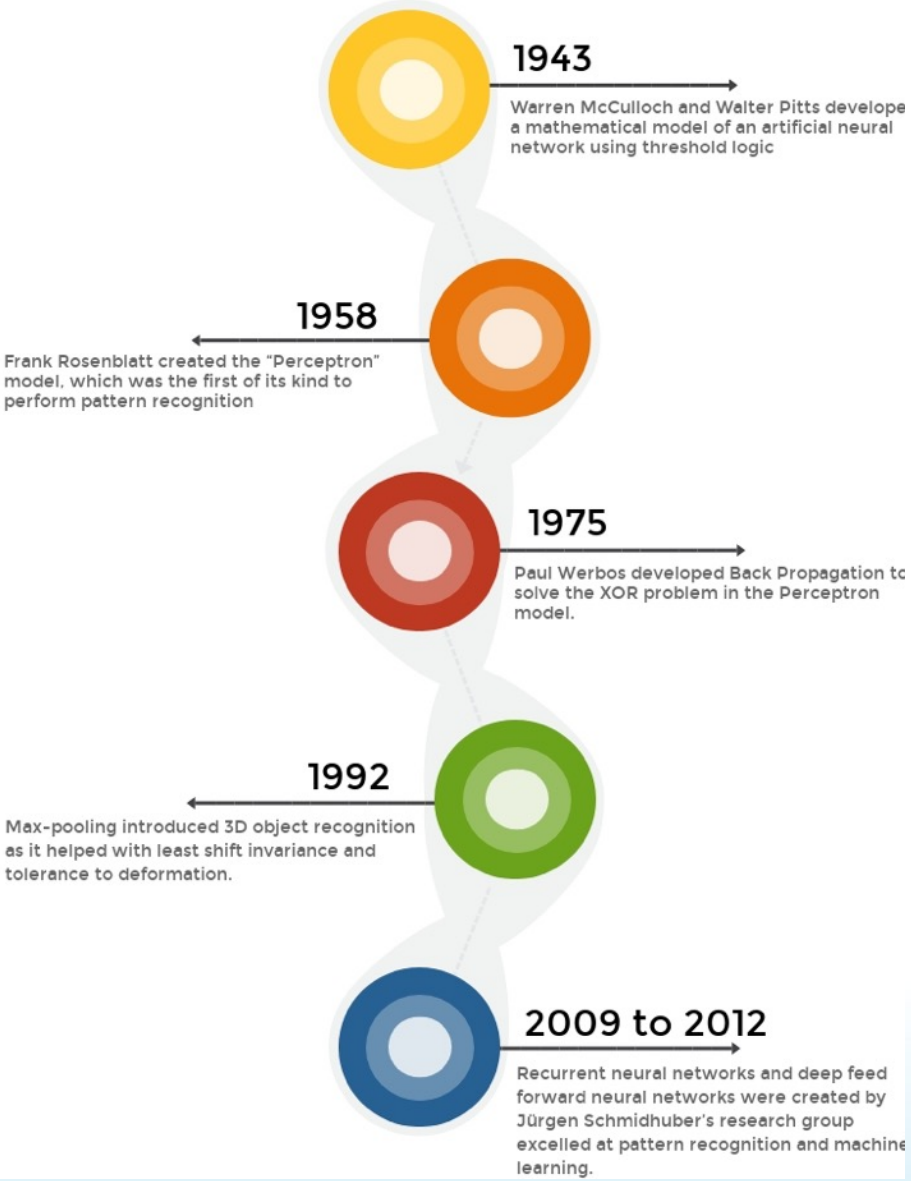
UNIVERSITY of WEST FLORIDA

- Introduction to linear classifiers
- Implementing perceptron
- Training adaptive linear neuron (Adaline) via gradient descent
- Hyperparameters of the perceptron and Adaline learning algorithms
- Feature scaling and gradient descent
- Full batch, stochastic and mini-batch gradient descent
- Code implementation in Jupyter notebook
- Summary and preview of next module

Artificial Neurons History



- The concept of artificial neurons was first proposed in 1943 by Warren McCulloch and Walter Pits in their publication of the McCulloch-Pitts (MCP) neuron model
- The MCP neuron functioned as a simple logic gate with binary inputs
- Signals arrived at the dendrites, where they were integrated in the cell body
- If the accumulated signal exceeded a certain threshold, it would be passed on by the axon
- In 1958, Frank Rosenblatt published the perceptron learning rule for training the MCP neuron model.



AN – Formal Def

UNIVERSITY of WEST FLORIDA

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m \qquad \sigma(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otehrwise} \end{cases}$$

Let's simplify this:

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

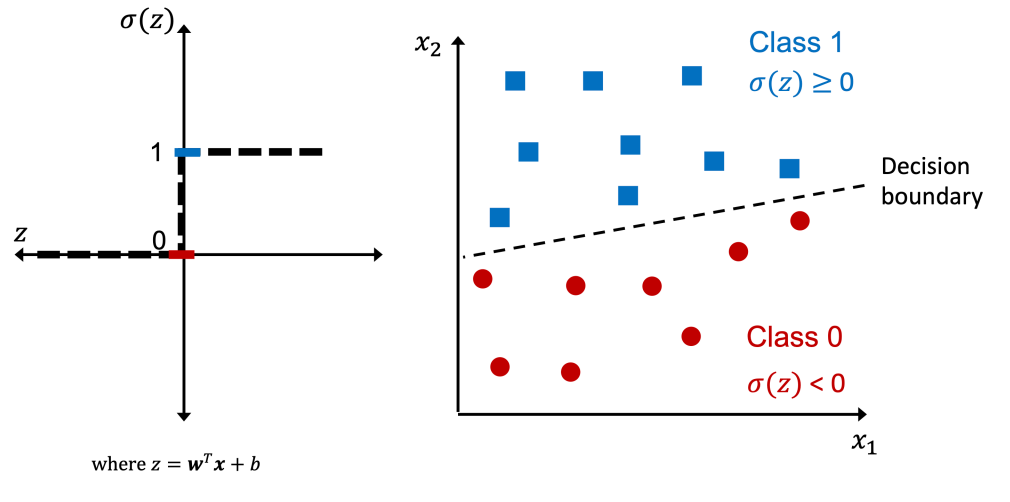
$$b = -\theta$$

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

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

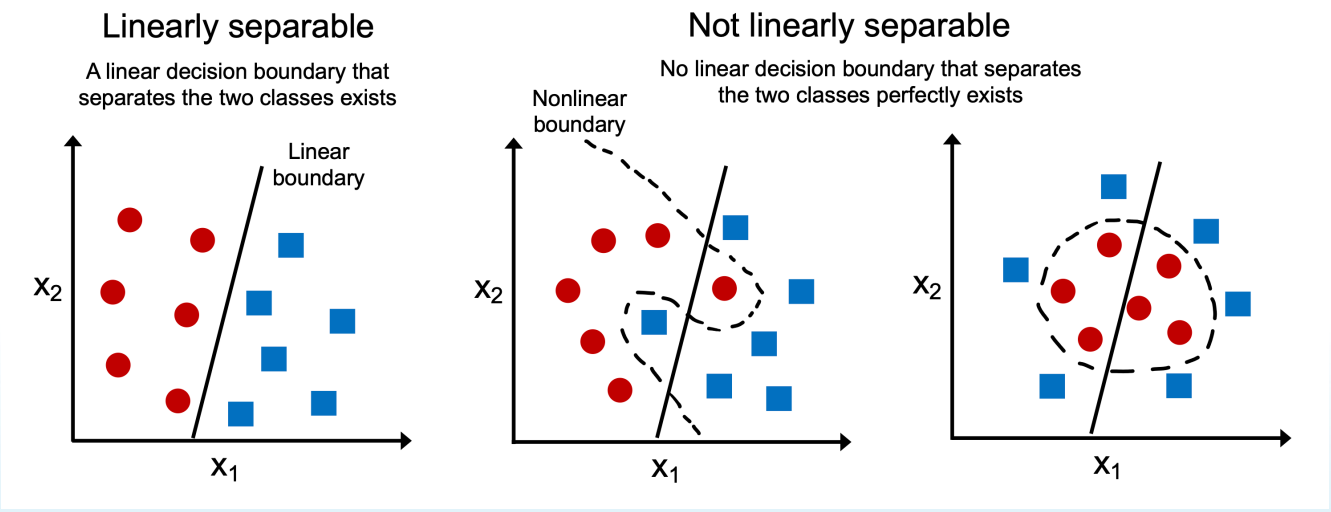
Decision Boundary for AN

- The decision boundary is created by the threshold function of the artificial neuron.
- The threshold function is a linear boundary that separates the input space into two regions.
- One region corresponds to one class and the other region corresponds to the other class.
- The decision boundary is determined by the weights and bias of the artificial neuron.
- The weights and bias can be adjusted during the training process to optimize the classification performance.
- In binary classification problems, the decision boundary is used to assign new input samples to one of the two classes.
- The decision boundary can also be used to estimate the probability of a sample belonging to one of the classes by considering the distance of the sample to the boundary.



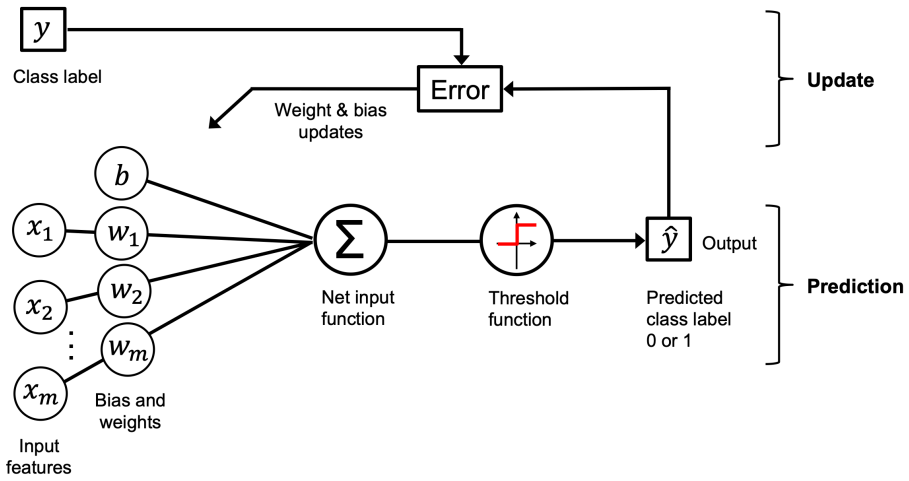
Perceptron Learning Rule

- The perceptron learning rule is a method for adjusting the weights of an artificial neuron to correctly classify a set of input-output pairs.
- The rule is based on the following update rule for the weights: $w_i \leftarrow w_i + \Delta w_i$
- Where $\Delta w_i = \eta(y - \hat{y})x_i$, with η being the learning rate, y the true output and $\hat{y}$ the predicted output of the neuro.
- This learning rule is guaranteed to converge for linearly separable data, but may not converge for non-linearly separable data.



Perceptron Algorithm for Training Artificial Neurons

- Initialize the weights & bias unit to 0 or small random numbers
- For each training example, $x^{(i)}$:
  - Compute the output value, $\hat{y}^{(i)}$
  - Update the weight & bias unit



$$w_i := w_j + \Delta w_j$$
$$b := b + \Delta b$$

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

η : learning rate, typically constant between 0.0 and 1.0

$y^{(i)}$: true class label of i-th training example

$\hat{y}^{(i)}$: predicted class label

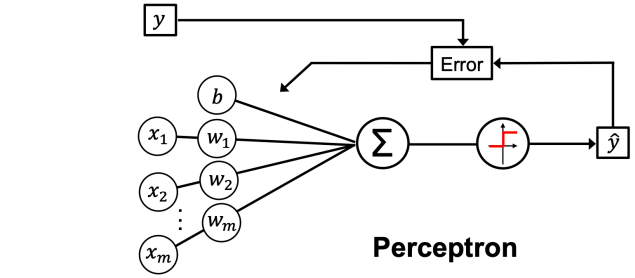
Exploring the Perceptron with Code

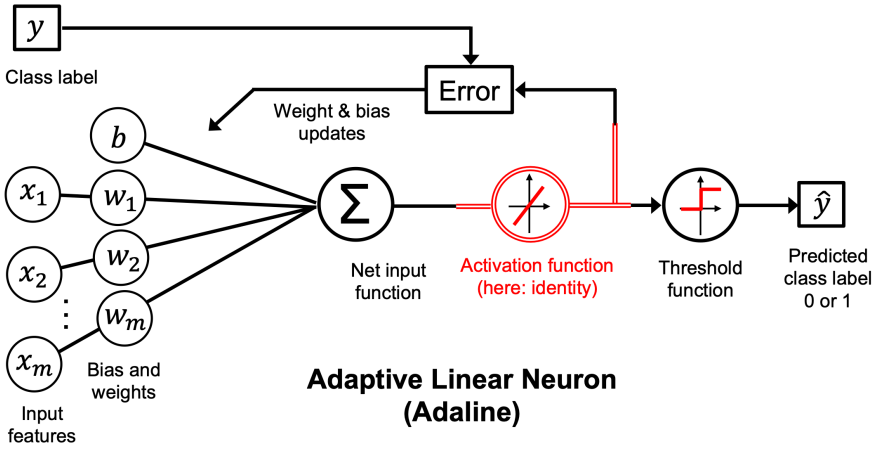
UNIVERSITY of WEST FLORIDA

- We have seen how the perceptron algorithm creates a linear decision boundary for binary classification problem.
- In this section, we will be implementing the perceptron algorithm in Python.
- Are you ready to see the perceptron in action?

Adaptive Linear Neurons (Adaline)

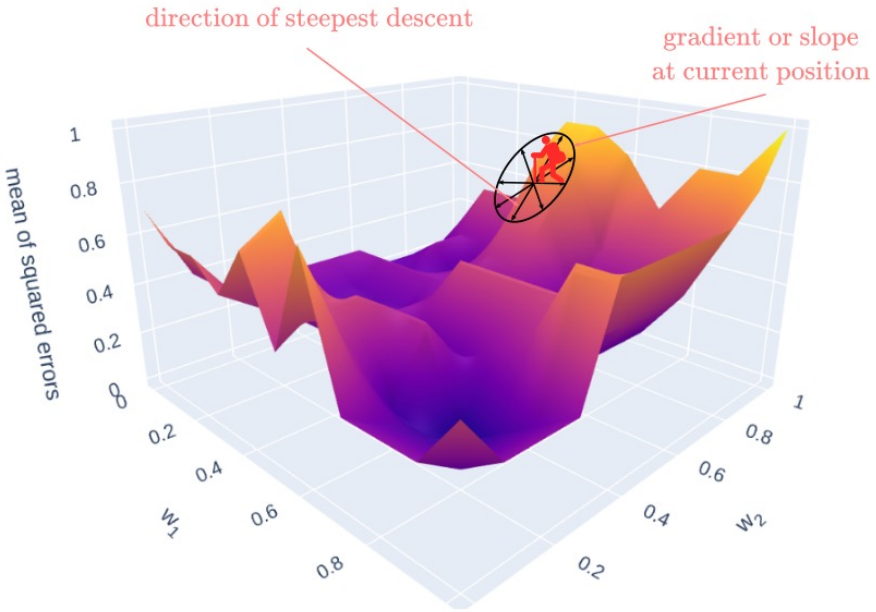
- Adaline is a type of single-layer neural network (NN)
- Developed by Bernard Widrow and Tedd Hoff in 1960, a few years after the perceptron algorithm
- Adaline uses a continuous loss function, which allows for more fine-grained weight updates during the learning process
- Adaline lays the foundation for understanding other ML algorithms for classification, such as logistic regression, support vector machines, multilayer neural networks, and linear regression





What's new in Adaline?

- Adaline uses a linear activation function ($\sigma(z) = z$) instead of a step function used in perceptron
- This allows Adaline to continuously update the weights during the learning process, rather than only making hard decisions
- Adaline also uses a different rule for weight update, known as the Widrow-Hoff rule, which is based on minimizing the cost function (often mean squared error)
- Adaline can also handle non-linearly separable data, unlike perceptron.



Minimizing Loss function with Gradient Descent

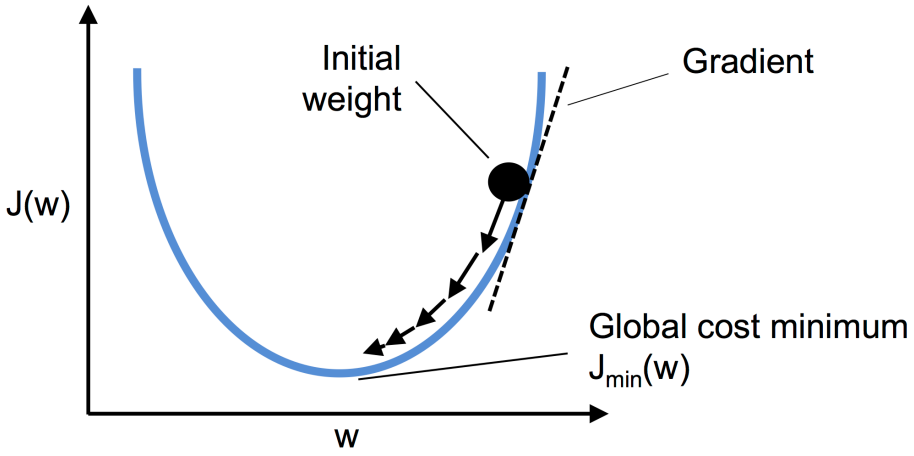
UNIVERSITY of WEST FLORIDA

- One of the key ingredients of Supervised ML is a defined objective function to be optimized during the learning process.
- The Adaline algorithm uses the mean squared error (MSE) as the loss function, L, which measures the difference between the calculated outcome and the true class label.
- $L(\boldsymbol{w}, b) = \frac{1}{2n}\Sigma_{i=1}^{n}\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2$ is the mathematical representation of the loss function, where n is the number of training examples and $y^{(i)}$ and $\sigma\left(z^{(i)}\right)$ are the true class label and the calculated outcome for the i-th training example respectively.
- The loss function is differentiable due to the linear activation function used in Adaline as opposed to the step function used in the perceptron algorithm.
- The loss function is also convex, which means that we can use a powerful optimization algorithm called gradient descent to find the weights that minimize the loss function.

Gradient Descent: Navigating the Loss Landscape

UNIVERSITY of WEST FLORIDA

- The goal of gradient descent is to find the minimum of a convex loss function
- The algorithm starts at a random point on the loss landscape and iteratively moves towards the global minimum
- In each iteration, the algorithm takes a step in the direction of the negative gradient, which is the direction of steepest descent
- The step size is determined by the value of the learning rate, which controls how fast the algorithm converges.



Updating Model Parameter

$$L(\boldsymbol{w}, b) = \frac{1}{2n}\Sigma_{i=1}^{n}\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2$$

The parameter changes, $\Delta\boldsymbol{w}$ and Δb, are defined as the negative gradient multiplied by the learning rate, η:

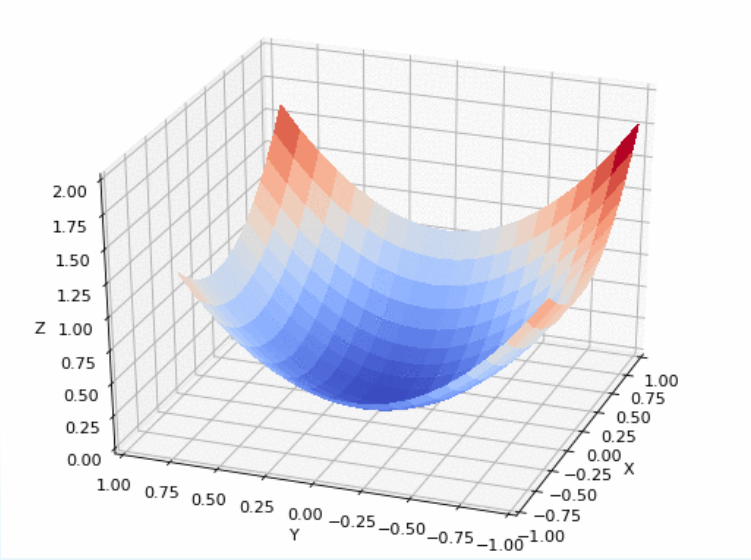
$$\boldsymbol{\Delta w} = -\eta\Delta_w L(\boldsymbol{w}, b), \qquad \frac{\partial L}{\partial w_j} = -\frac{2}{n}\Sigma_i\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)x_j^{(i)} \qquad \Delta w_j = -\eta\frac{\partial L}{\partial w_j}$$

$$\Delta b = -\eta\Delta_b L(\boldsymbol{w}, b) \qquad \frac{\partial L}{\partial b} = -\frac{2}{n}\Sigma_i\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right) \qquad \Delta b = -\eta\frac{\partial L}{\partial b}$$

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

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

$$b := b + \Delta b$$



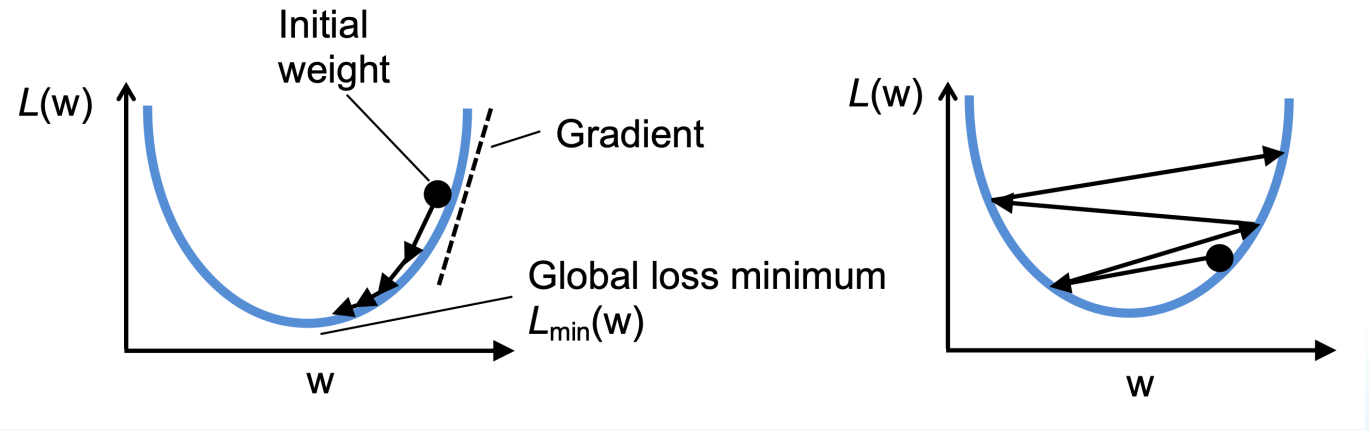
Implementing Adaline in Code

UNIVERSITY of WEST FLORIDA

- Defining the Adaline class with the necessary attributes and methods, such as the “fit” method for training the model and the “predict” method for making predictions
- Using the Adaline class to fit the model to the training data and make predictions on new data
- Visualizing the performance of the model through plots of the cost function over time and decision regions

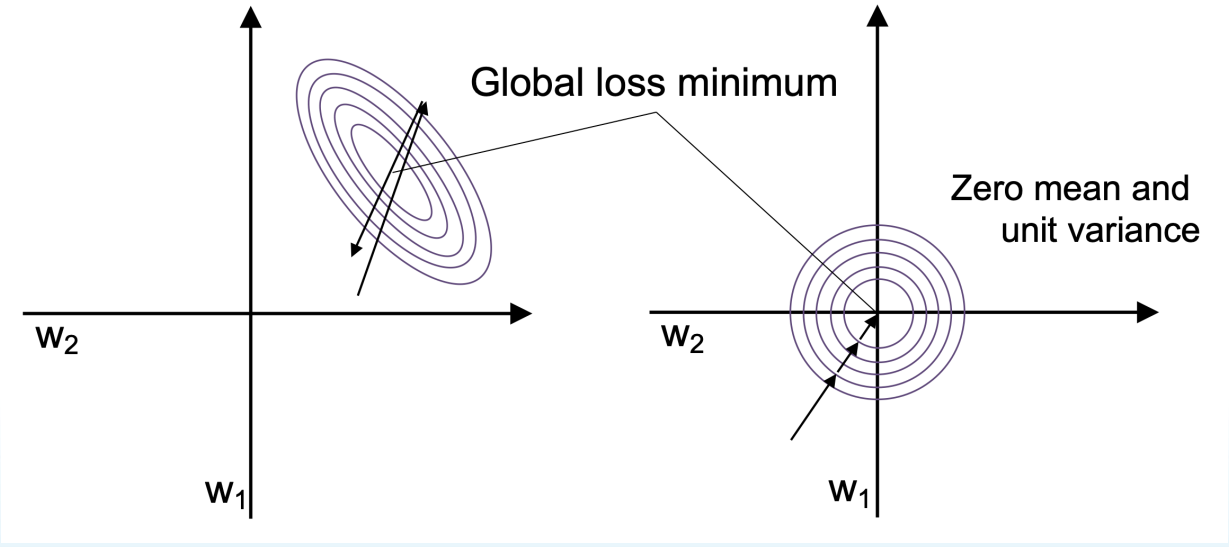
Hyperparameters

- Hyperparameters are parameters that are not learned from the data, but are set by the user.
- In the Adaline algorithm, the learning rate and the number of epochs are examples of hyperparameters.
- The learning rate controls the step size of the weight updates, and a higher learning rate can lead to faster convergence, but also the risk of overshooting the global minimum.
- A smaller learning rate may require more epochs to converge, but it can reduce the risk of overshooting the global minimum.
- The number of epochs controls the number of times the learning algorithm will work through the entire training dataset.
- A higher number of epochs can lead to better convergence, but it also increases the risk of overfitting.



Feature Scaling

- Gradient descent is one of the many ML algorithms that benefits from feature scaling
- One feature scaling method is standardization
- Normalization procedure helps gradient descent to converge faster
- Procedure: shift the mean of each feature so that is centered around zero and has unit variance
- Ex. Standardizing jth feature: $x_j' = \frac{x_j - \mu_j}{\sigma_j}$
- Comparison of the unsclaed and standarized features on gradients update is shown in the picture on the slide



Full batch gradient descent

UNIVERSITY of WEST FLORIDA

- Full batch gradient descent is a method of finding the global minimum of a loss function by taking steps in the direction of the gradient, calculated over the entire training set.
- However, it can be computationally expensive as it requires re-evaluating the entire training dataset at each step.
- An alternative is stochastic gradient descent (SGD), also known as iterative or online gradient descent, which utilizes a small subset of the training data to update the parameters at each step.

Stochastic Gradient Descent

UNIVERSITY of WEST FLORIDA

- Stochastic gradient descent (SGD) is a variant of gradient descent where the weights are updated incrementally for each training example, rather than based on the sum of the accumulated error over all training examples.
- The update rule for the weight w_j and bias b in SGD is:
  - $\Delta w_j = \eta\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right) x_j^{(i)}, w_i := w_j + \Delta w_j$
  - $\Delta b = \eta\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right), b := b + \Delta b$
- One of the benefits of SGD is that it can be used for online learning and is computationally more efficient than full batch gradient descent

SGD properties:

UNIVERSITY of WEST FLORIDA

- SGD is an approximation of the full-batch gradient descent algorithm
- It typically converges faster due to the frequent updates made to the model parameters
- The error surface is much noisier in SGD compared to full-batch gradient descent
- SGD can escape local minima more readily when working with non-linear loss functions
- The training dataset should be shuffled every epoch to avoid cycles in the updates
- SGD is commonly used in online learning, where the model is trained on the fly as new data arrives
- It is useful when working with large amounts of data (such as customer data in a web application) and allows the system to immediately adapt to changes and discard the training data after updating the model

Mini-batch gradient descent

UNIVERSITY of WEST FLORIDA

- Mini-batch gradient descent is a compromise between full batch gradient descent and stochastic gradient descent.
- It applies full batch gradient descent to smaller subsets of the training data, such as 32 training examples at a time.
- It converges faster than full batch gradient descent and allows for more computational efficiency by using vectorized operations to replace the "for loop" over the training examples used in SGD.

Code Implementation

UNIVERSITY of WEST FLORIDA

- Transition to Jupyter notebook for code implementation
- Explanation of how the concepts discussed in the previous slides can be applied in practice using the Jupyter notebook
- Preview of the code and examples that will be covered in the next section of the lecture

Summary

UNIVERSITY of WEST FLORIDA

- Summary of key concepts covered in this module:
  - Linear classifiers and their basic concepts
  - Implementation of the perceptron algorithm
  - Training of the Adaline algorithm using gradient descent and online learning via stochastic gradient descent
- Preview of next module:
  - Overview of machine learning classifiers using scikit-learn
  - Hands-on implementation and practical applications of various classifiers