Deep Learning

Lesson 3—How to Train an Artificial Neural Network

Learning Objectives

- Explore the layers of an Artificial Neural Network(ANN).
- Understand how ANN is trained using Perceptron learning rule.
- Explain the implementation of Adaline rule in training ANN.
- Describe the process of minimizing cost functions using Gradient Descent rule.
- Analyze how learning rate is tuned to converge an ANN.

How to Train an Artificial Neural Network

Topic 1—Introduction

Artificial Neural Networks (ANN)

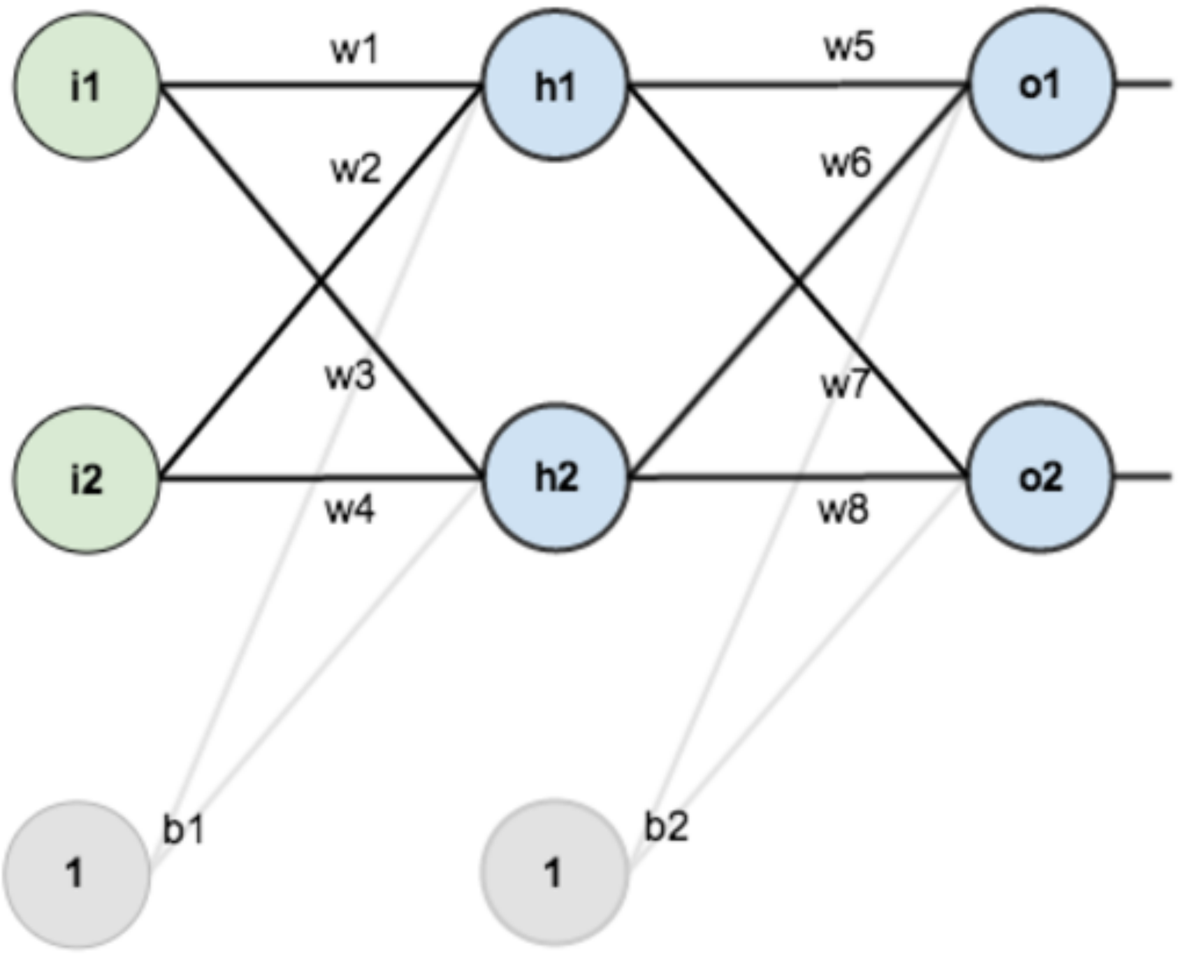
DEFINITION

> “Artificial Neural Network is a computing system made up of a number of simple, highly interconnected processing elements which process information by their dynamic state response to external inputs.”
>
> \- Robert Hecht-Nielsen

Layers of ANN

Input for hidden layer

Output for hidden layer



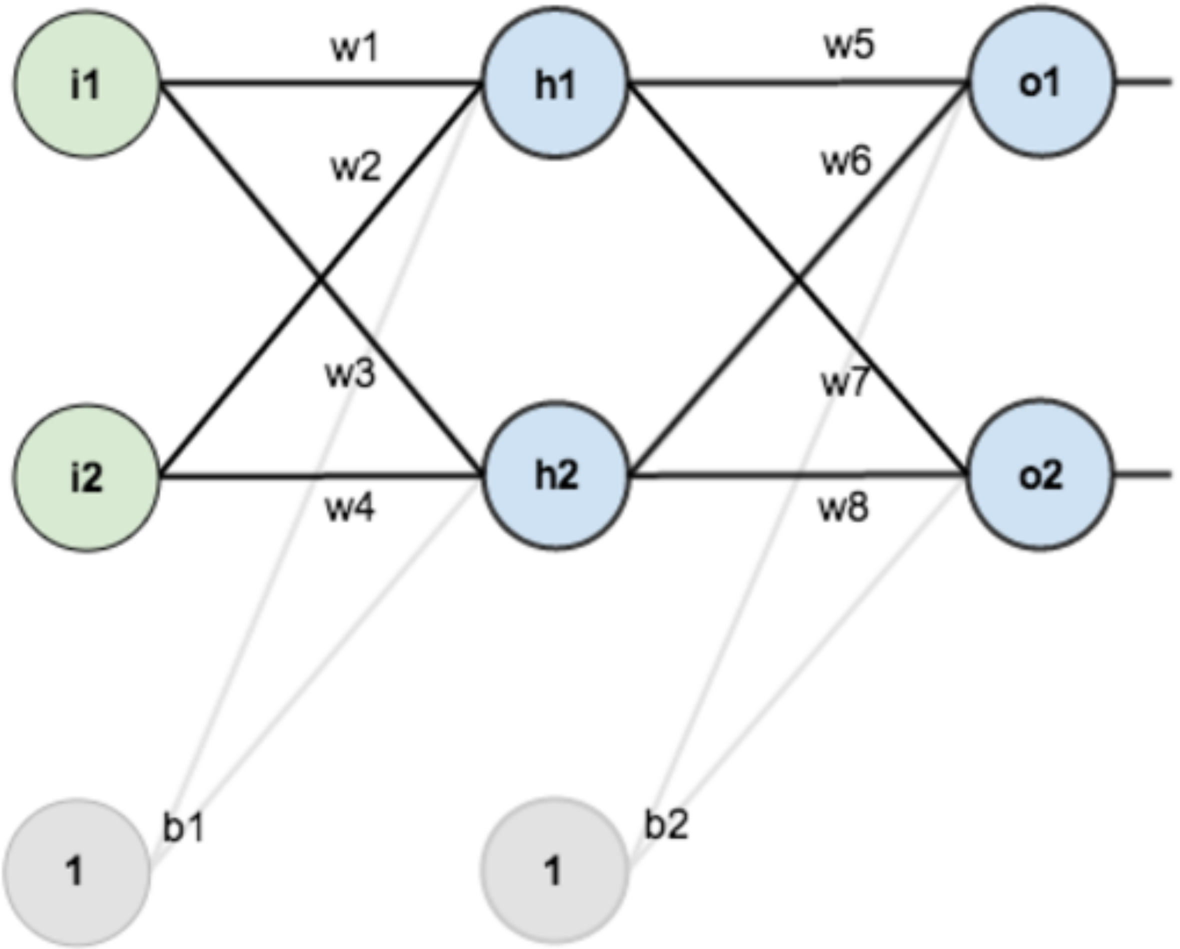
- The diagram shows a three layered neural network:
  - Input layer
  - Output layer
  - One hidden layer.

- The input for hidden layer neuron is weighted outputs of input neurons plus a bias term.

Total net input for h_1

$$net_{h1} = w1*i1 + w2*i2 + b1*1$$

Layers of ANN



Input for hidden layer

Output for hidden layer

- The output of hidden layer passes through a sigmoid transformation using the sigmoid activation function.

Output of h_1 :-
(Apply sigmoid activation function)

$out_{h1} = 1/(1+e^{-neth1})$

- The output of a sigmoid is a a value between 0 and 1.
- Finally the outputs of the hidden layer are again weighted to produce the output layer values o1 and o2.

How to Train ANN

- Single layer neural network(or perceptrons) can be trained using either the Perceptron training rule or the Adaline rule.

Perceptron Training Rule (Rosenblatt's Rule)

- Works well when training samples are linearly separable
- Updates weights based on error in the threshold perceptron output (e.g.: +1 and -1)

ADaptive LInear NEuron (Adaline) Rule (Widrow-Hoff Rule)

- Works well even when the training samples are not linearly separable
- Makes necessary changes to the template
- Updates weights based on the error in non-threshold linear combination of outputs
- Converges towards best-fit approximation of the target output
- Provides basis for backpropagation algorithm, which can learn networks with many interconnected units

How to Train an Artificial Neural Network

Topic 2—Perceptron Learning Rule

Perceptron Learning Rule (Rosenblatt's Rule)

DEFINITION OF PERCEPTRON

> A perceptron is a computational unit that calculates output based on weighted input parameters.

Perceptron Learning Rule

STEPS TO FOLLOW

- The basic idea is to mimic how a single neuron in the brain works: it either fires or it doesn't.
- Rosenblatt's initial perceptron rule is fairly simple and can be summarized by the following steps:

01 Initialize the weights to 0 or small random numbers.

02 For each training sample x(i) : Compute the output value $\hat{y}$.

03 Update the weights.

- Here the output value is the predicted class label produced by the unit step function.

Perceptron Learning Rule (Contd.)

- Weight adjustment at each step is written as:

$$w_j := w_j + \Delta w_j$$

- The value of weight change is calculated as:
  - Here "η" is the learning rate (typically a constant between 0.0 and 1.0)
  - "y(i)" is the true class label
  - "$\hat{y}$(i)" is the predicted class label

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

- It is important to note that all weights in the weight vector are being updated simultaneously. Hence for a two dimensional dataset, the weight update would be:

$$\Delta w_0 = \eta\left(y^{(i)} - output^{(i)}\right)$$

$$\Delta w_1 = \eta\left(y^{(i)} - output^{(i)}\right)x_1^{(i)}$$

$$\Delta w_2 = \eta\left(y^{(i)} - output^{(i)}\right)x_2^{(i)}$$

Perceptron Learning Rule (Contd.)

PREDICTION OF THE CLASS LABEL

Case 1: Perceptron predicts the class label correctly.

- The weights remain unchanged.

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

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

Case 2: Perceptron predicts the class label wrongly.

- The weights are being pushed towards the direction of the positive or negative target class.

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

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

Perceptron Learning Rule

CONVERGENCE IN NEURAL NETWORK

- Convergence is performed so that cost function gets minimized and preferably reaches the global minima. It is also done to find the best possible weights to minimize the classification problem.

- Convergence of the learning algorithms is guaranteed only if:
  - The two classes are linearly separable
  - The learning rate is sufficiently small

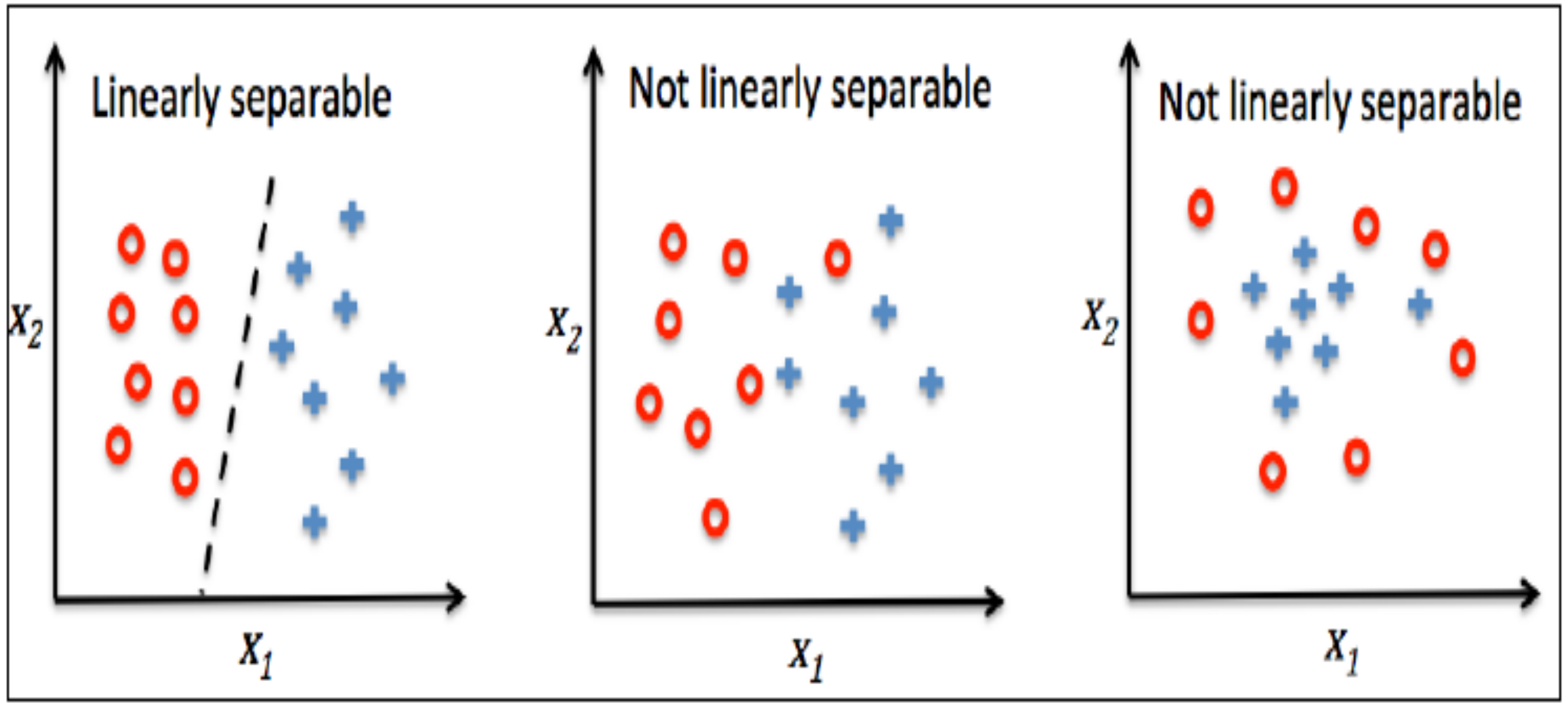


Image Source : "Python Machine Learning" by Sebastian Raschka

Perceptron Learning Rule (Contd.)

CONVERGENCE IN NEURAL NETWORK

NOTE:

If the two classes can't be separated by a linear decision boundary, you can set a maximum number of passes over the training dataset (epochs) and/or a threshold for the number of tolerated misclassifications.

The perceptron would never stop updating the weights otherwise.

Image Source : "Python Machine Learning" by Sebastian Raschka

Perceptron Learning Rule

SUMMARY

The perceptron receives the inputs of a sample x and combines them with the weights w to compute the net input.

The net input is then passed on to the threshold function, which generates a binary output -1 or +1: the predicted class label of the sample.

During the learning phase, this output is used to calculate the error of the prediction and update the weights.

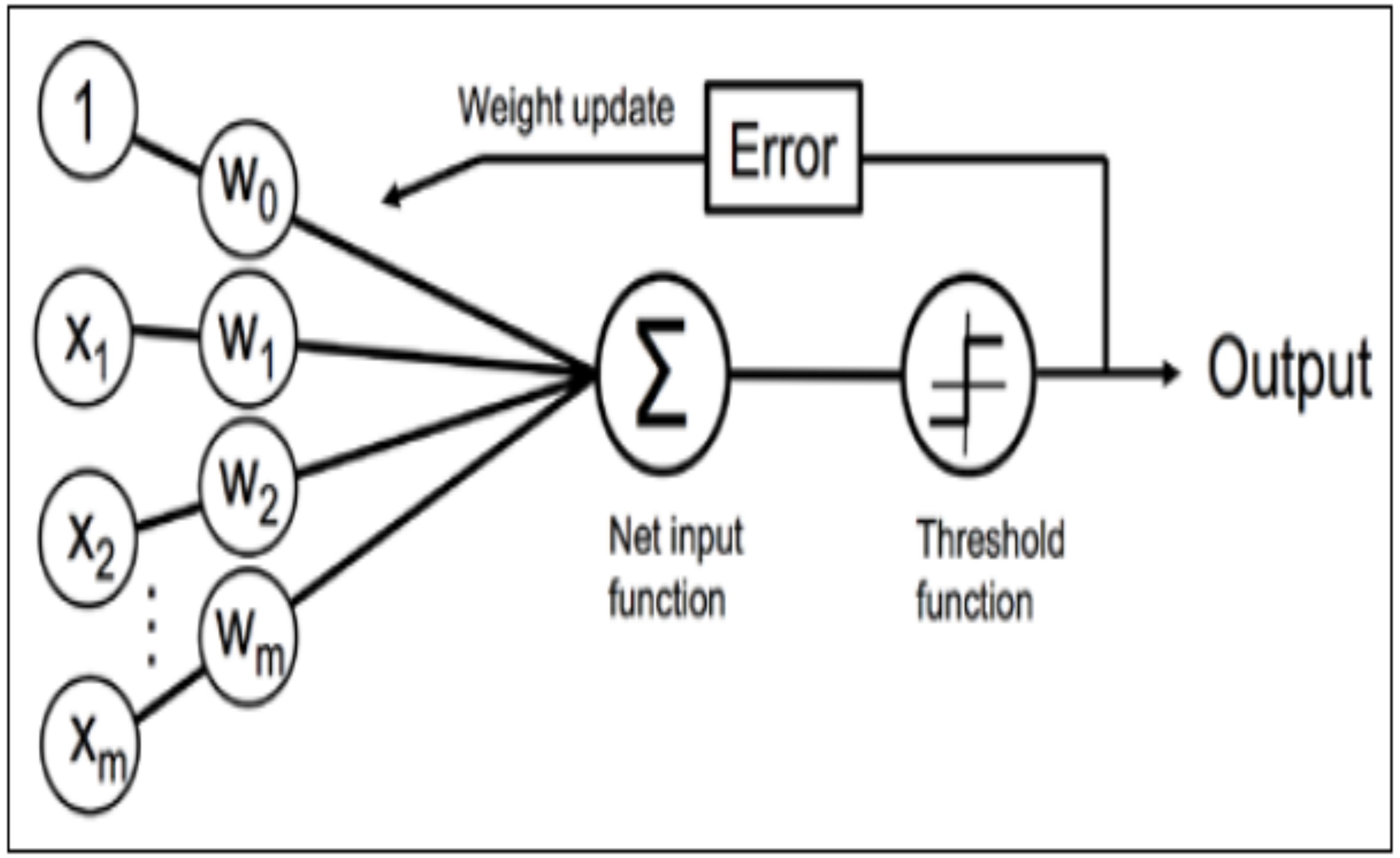


Image Source : "Python Machine Learning" by Sebastian Raschka

simplilearn

How to Train an Artificial Neural Network

Topic 3—Adaline rule

Adaline Rule (Widrow-Hoff Rule)

- In Adaline, the weights are updated based on a linear activation function.
- The linear activation function $\varphi(z)$ is the identity function of the net input, so that:

$$\varphi(w^T x) = w^T x$$

- While the linear activation function is used for learning the weights, a threshold function is used to make the final prediction, which is similar to the unit step function.

Adaline Rule Vs Perceptron Rule

Adaline Rule

- Weights are updated based on a linear activation function.
- Compares the true class labels with the linear activation function's continuous valued output to compute the model error and update the weights.

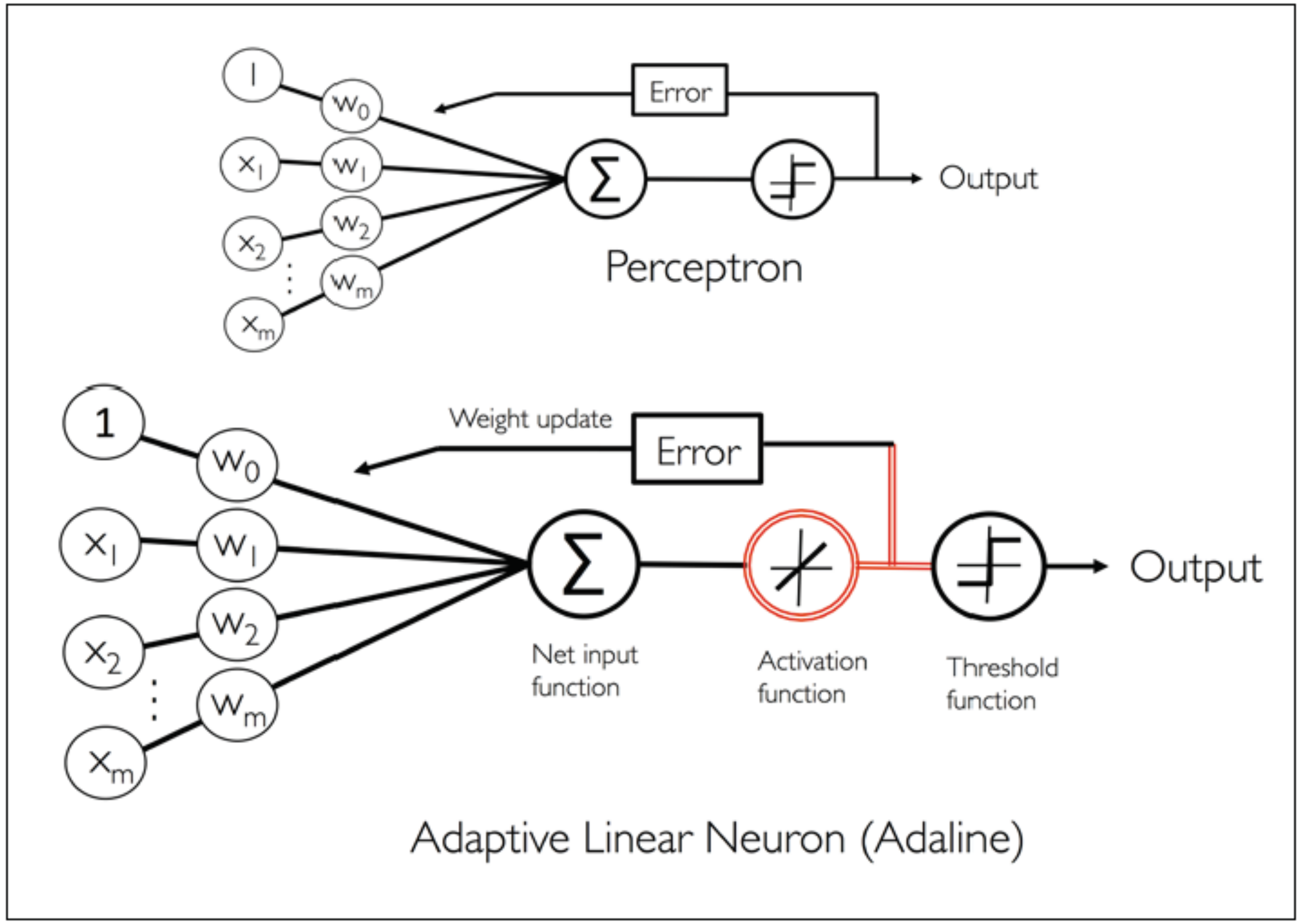


Image Source : "Python Machine Learning" by Sebastian Raschka

Perceptron Rule

- Weights are updated based on a unit step function.
- Compares the true class labels with the predicted class labels to compute the model error and update the weights.

Adaline Rule

MINIMIZING COST FUNCTIONS

- The most common neural networks belong to supervised learning category, where ground truth output labels are available for training data.
- One key technique in supervised learning is to optimize an objective function, which enables the learning process.
- This objective function is often a cost function which is to be minimized.

Minimizing Cost Functions

SUM OF SQUARED ERRORS(SSE)

- In Adaline, Sum of Squared Errors (SSE) is the cost function J which needs to be minimized.

- SSE is squared difference of calculated outcomes and true class labels.

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

- Minimizing this brings the predicted output close to ground truth labels. Also, squaring makes it differentiable.

Minimizing Cost Functions (Contd.)

- The main advantage of this continuous linear activation function, in contrast to the unit step function, is that the cost function becomes differentiable and convex.

- Hence, a simple yet powerful optimization algorithm called Gradient Descent can be used to find the weights that minimize the cost function to classify the samples in the dataset.

How to Train an Artificial Neural Network

Topic 4—Use of Gradient Descent in Adaline rule

Minimizing Cost Functions With Gradient Descent

The main idea behind gradient descent is to go down the hill of cost function until a local or global minimum point is reached.

In each iteration, a step is taken in the opposite direction of the gradient where the step size is determined by the value of the learning rate.

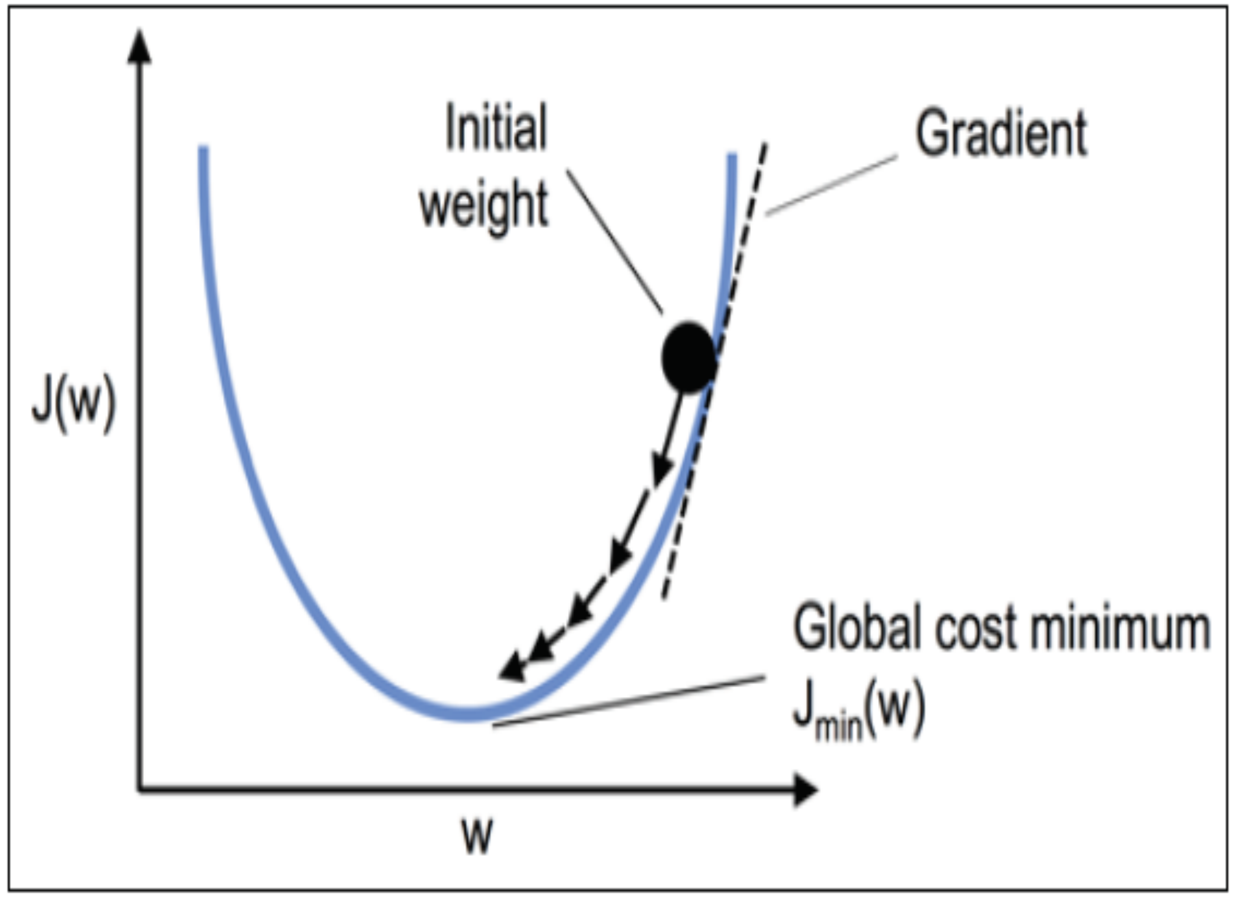


Image Source : "Python Machine Learning" by Sebastian Raschka

Steps to Minimize Cost Functions With Gradient Descent

Using gradient descent, update weight by taking a step in the opposite direction of the gradient.

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

The weight change "Δw" is defined as negative gradient multiplied by the learning rate "η".

$$\Delta \boldsymbol{w} = -\eta \nabla J(\boldsymbol{w})$$

To compute the gradient of the cost function, compute the partial derivative of the cost function with respect to each weight "w_j". So it can be written as:

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

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

Since all weights are updated simultaneously, the Adaline learning rule becomes:

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

Steps to Minimize Cost Functions With Gradient Descent (Contd.)

The partial derivative of the SSE cost function with respect to the j[th] weight can be obtained as shown:

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

$$= \frac{1}{2} \frac{\partial}{\partial w_j} \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right)^2$$

$$= \frac{1}{2} \sum_i 2 \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - \phi\left(z^{(i)}\right) \right)$$

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

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

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

Difference Between Perceptron and Gradient Descent Rule

- Both perceptron and gradient descent seem to use the rule: $\Delta w_i = \eta(t - o)x_i$
- But in reality the rules are different:

Perceptron Rule	Gradient Descent Rule
o refers to the threshold output $o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$ The threshold output is not differentiable.	o refers to the linear unit output $o(\vec{x}) = \vec{w} \cdot \vec{x}$ The non-threshold output is differentiable.

A logistic regression model (core Machine Learning) is closely related to Adaline with the only difference being its activation and cost function.

Stochastic Gradient Descent (Incremental Gradient Descent)

- Issues with Gradient descent:
  - Converging to local minima is very slow.(thousands of gradient descent steps needed)
  - In case of multiple local minima, global minima may not be found.

- These issues can be alleviated with stochastic gradient descent:
  - In this, weights are updated incrementally, after error calculation for each sample d, rather than computing weight updates after summing errors over all samples of D.
  - In case of multiple local minima, stochastic gradient descent is a better choice to find the global minimum.

Batch mode Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode Gradient Descent:
Do until satisfied

- For each training example d in D
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

How to Train an Artificial Neural Network

Topic 5—Tune the Learning Rate

Tune the Learning Rate

HYPERPARAMETERS

Hyperparameters are parameters set by the data scientist / developer while building the model based on experience or by hit and trial.

- These parameters are not among those (unlike weights and biases) that get learnt during training.
- The hyperparameters of the perceptron and Adaline learning algorithms are:
  - Learning rate "η" (eta) and
  - Number of epochs (n_iter)
- The learning rate indicates the speed of learning, or a factor to moderate the rate of weight adjustment over multiple training loops.
- An Epoch refers to one complete training pass or one pass of the training loop.

Tune the Learning Rate

OPTIMAL CONVERGENCE

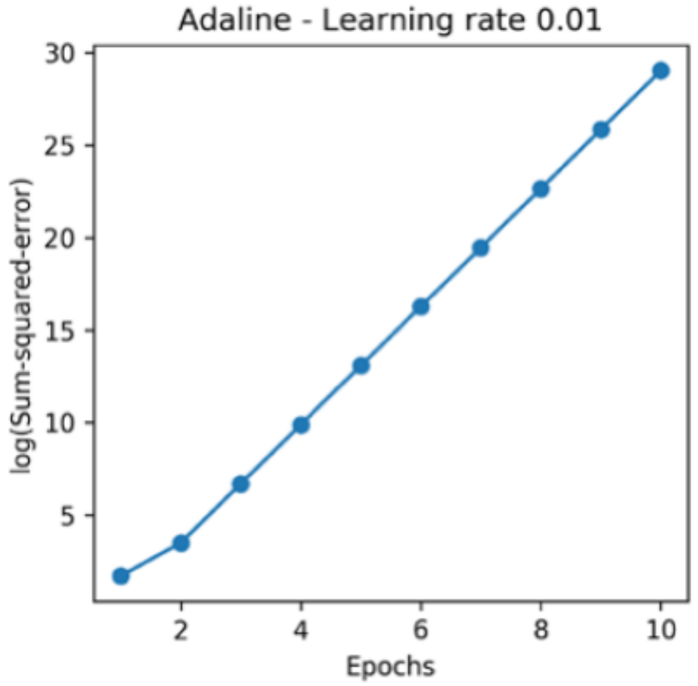
- In practice, it often requires some experimentation to find a good learning rate η for optimal convergence.

- So, let's choose two different learning rates, $\eta = 0.1$ and $\eta = 0.0001$, to start with.

- Plot the cost functions versus the number of epochs to see how well the Adaline implementation learns from the training data.

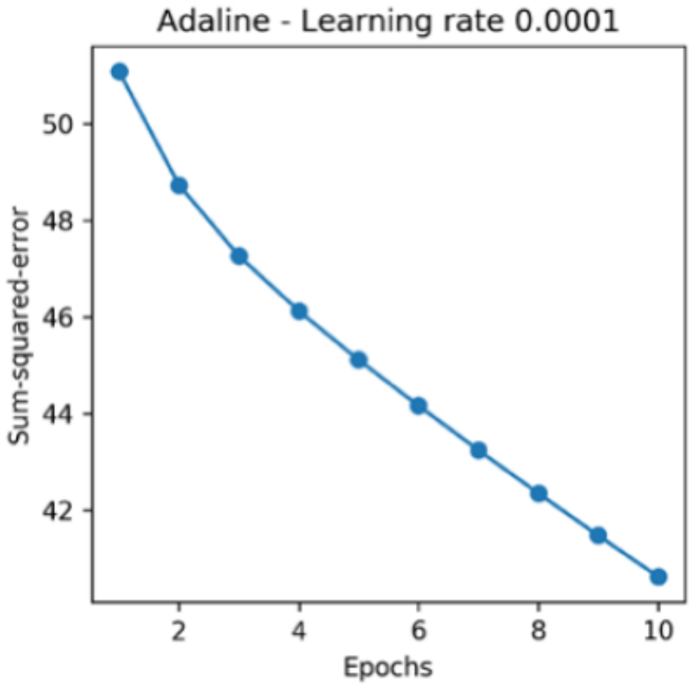
Tune the Learning Rate

Two different types of problem are encountered.



$\eta = 0.1$

This chart shows what could happen if a learning rate that is too large is chosen. Instead of minimizing the cost function, the error becomes larger in every epoch, because we overshoot the global minimum.



$\eta = 0.0001$

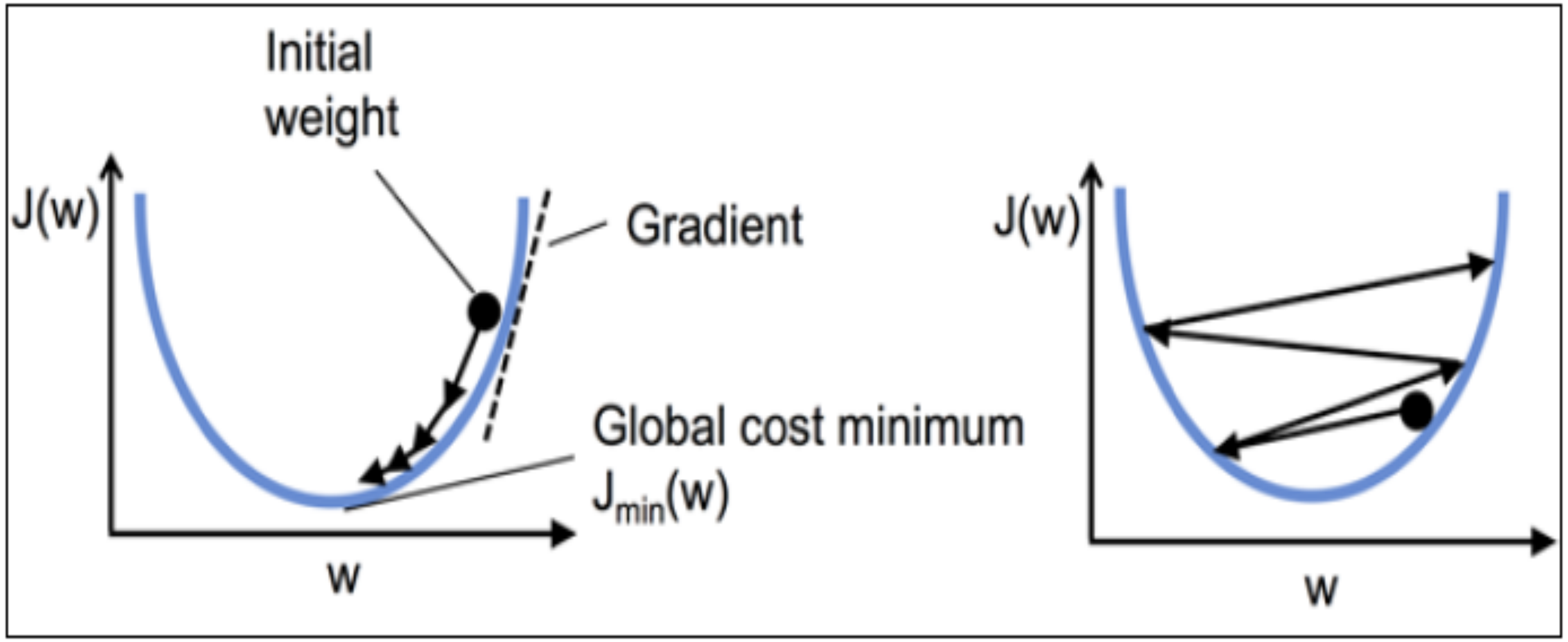
In this chart it is seen that the cost decreases, but the chosen learning rate is so small that the algorithm would require a very large number of epochs to converge to the global cost minimum.

Image Source : "Python Machine Learning" by Sebastian Raschka

Tune the Learning Rate

CONVERGENCE

The figure on the left demonstrates optimum learning rate, where the cost function converges to a global minimum.



The figure on the right shows a large learning rate, and the global minimum gets missed during weight adjustments.

Image Source : "Python Machine Learning" by Sebastian Raschka

Key Takeaways

- Artificial neural network has an input, output and a hidden layer. The output of the hidden layer is obtained by applying the sigmoid or some other activation function.
- Perceptron learning rule is best suited when the learning samples are linearly separable. The weights are updated based on a unit step function.
- In Adaline rule, weights are updated based on a linear activation function. They can be used even when the learning samples are not linearly separable.
- Gradient descent rule, often used as part of Adaline algorithm, can be used to find the weights that minimize the cost function.
- Selecting a large learning rate causes a large error and the global minimum gets missed during weight adjustments. Smaller learning rate ensures that the cost function converges to a global minimum.

This concludes “How to Train an Artificial Neural Network”

The next lesson is “Multilayer ANN.”