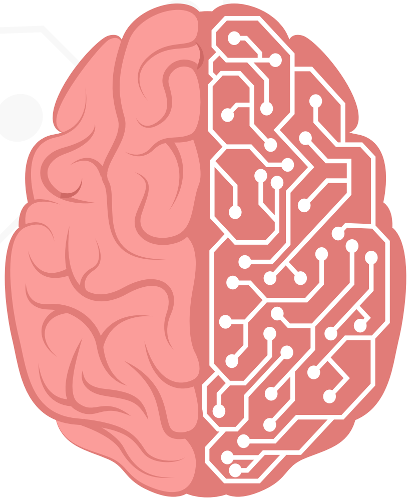


Neural Networks

2022-2023

LAB 4

Agenda

1. Adaline Algorithm
2. Task 2

Adaline Algorithm

- Stands for “*Adaptive Linear Neuron*”.
- This is the point where Adaline really deviates from Perceptron because there is no cost function in Perceptron and without a cost function Perceptron is minimizing nothing.
- This is why Adaline is considered as an improvement of Perceptron as you can see how it is improving its prediction by simply looking at its cost function.
- Gradient Descent:
  It is a first order iterative optimization algorithm for finding a local minimum of differential function.

Adaline Algorithm

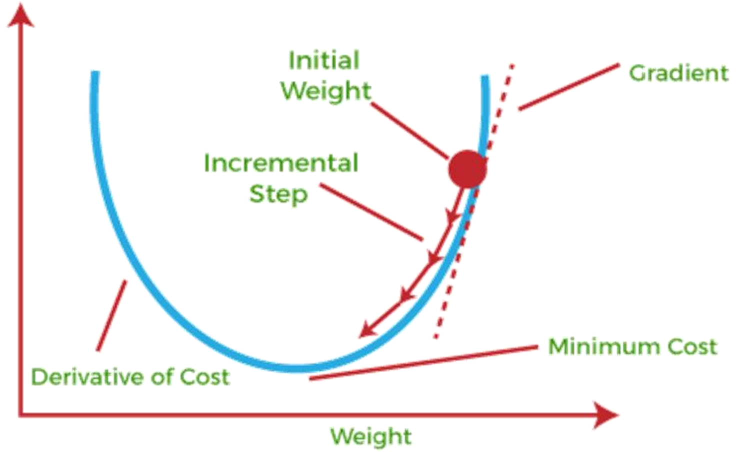
- Stands for "*Adaptive Linear Neuron*".
- The *Adaline* deviates from *Perceptron* because there is no cost function in *Perceptron* and without a cost function *Perceptron* is minimizing nothing.

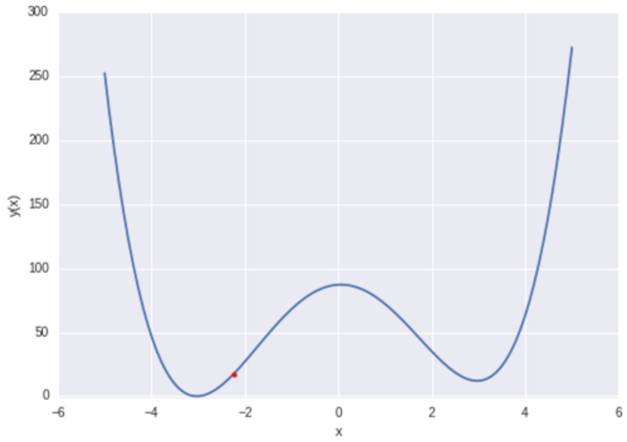
- The cost function in the **Adaline Algorithm** is mean square error.
- To minimize cost function, gradient descent is needed.
- The gradient decent should work with differentiable function (has derivative) as the weight update is on the slope's opposite direction

Gradient Decent

- It is a first-order optimization algorithm. To find a local minimum of a function, one takes a step proportional to the negative of the gradient of the function at the current point.

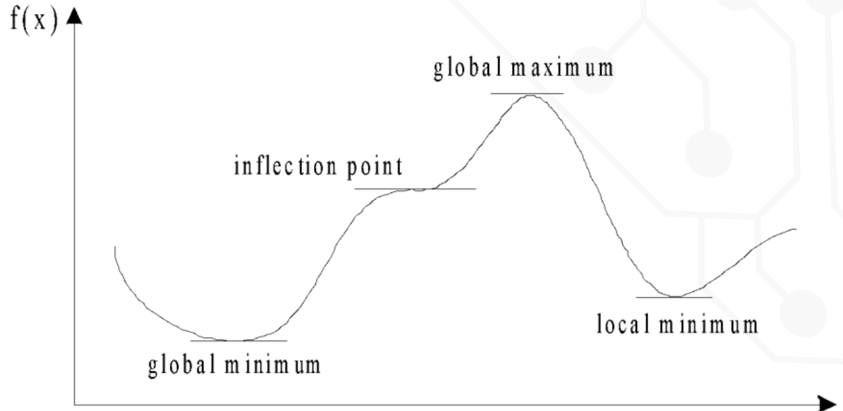
- If this step size, Learning rate, is too large, we will overshoot the minimum, that is, we won't even be able land at the minimum. If learning rate is too small, we will take too many iterations to get to the minimum. So, learning rate needs to be just right.





Gradient Decent

- Gradient is the slope of a function.
- Optimization is finding the “best” value of a function which is the minimum value of the function.
- The number of “turning points” of a function depend on the order of the function.
- Not all turning points are minimum.
- The least of all the minimum points is called the “global” minimum.
- Every minimum is a “local” minimum
- We use the gradient descent to find which weights and biases minimize the cost function



Mean Square Error (Cost function)

- 1. **MSE** measures the average squared difference between the desired values and what is estimated.

- If **O** is a vector of predictions generated from a sample of *n* data points on all variables, and **T** is the vector of observed values of the variable being predicted, then the within-sample MSE of the predictor is computed as

$$\mathbf{MSE(n)} = \frac{1}{n}\sum_{i=1}^{n}(t_i - o_i)^2$$

Mean Square Error Derivative (Gradient)

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

Adaline learning algorithm

```
1. 1. Start with a randomly chosen weight vector w(0)
2.    Repeat
3.        For each training vector pair $(x_i, t_i)$
4.            Evaluate the output $y_i$ when $x_i$ is the input according to
                  $y_i = [\, w(i)^T . x(i) \,]$
5.            Compute the error => $e = t_i - y_i$
6.            Update the weight => $w_{i+1} = w_i + \eta . e . x_i$
          End for
   until the stopping criteria is reached by find the Mean square error across all the training samples.
```

Calculate the error for each sample (after each epoch) using the updated weights then calculate the MSE.

$$\text{MSE}(n) = \frac{1}{m} \sum_{k=1}^{m} \frac{1}{2} e(n)^2$$

Stopping criteria:

if the Mean Squared Error **across all the training** samples is less than a specified value, stop the

What is in common ?

- What is common between the Perceptron and the Adaline algorithms?

    1. They are classifiers for binary classification
    2. Both have a linear decision boundary
    3. Both can learn iteratively, sample by sample (the Perceptron naturally, and Adaline via stochastic gradient descent)
    4. Both use a threshold function

What is different ?

- What is the difference between the Perceptron and the Adaline algorithms?

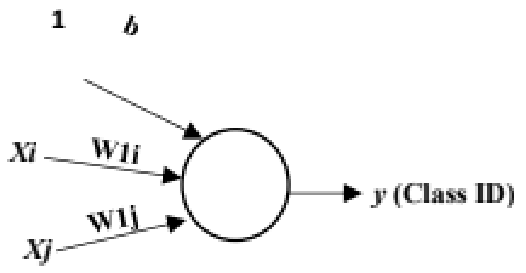
  1. The Perceptron uses the class labels to learn model coefficients
  2. Adaline uses continuous predicted values (from the net input) to learn the model coefficients, which is more “powerful” since it tells us by “how much” we were right or wrong
  3. The Adaline, is similar to the perceptron, but their transfer function is linear rather than hard limiting. This allows their output to take on any value.
  4. Perceptron fixes binary error; ADALINE minimizes continuous error.

Task 2 - GUI

1. **User Input:**
- Select two features
- Select two classes (C1 & C2 or C1 & C3 or C2 & C3)
- Enter learning rate (eta)
- Enter number of epochs (m)
- Enter MSE threshold (mse_threshold)
- Add bias or not (Checkbox)

2. **Initialization:**
- Number of features = 2.
- Number of classes = 2.
- Weights + Bias = small random numbers

3. **Classification:**
- Sample (single sample to be classified).



Task 2 - Description

1. **Implement the Adaline learning algorithm using MSE**

- Single layer neural networks which can be able to classify a stream of input data to one of a set of predefined classes.

- Use the penguins data in both your training and testing processes. (Each class has 50 samples: train NN with 30 non-repeated samples randomly selected, and test it with the remaining 20 samples)

Task 2 - Description

2. After training

- Draw a line that can discriminate between the two learned classes.

- Test the classifier with the remaining 20 samples of each selected classes and find confusion matrix and compute overall accuracy.

Task 2 - Workflow

- **Training Phase: (repeat the following *m* epochs)**

  Assuming that we have ***n*** training samples $\{sample_i : i = 1 \rightarrow \boldsymbol{n}\}$
  - Fetch features (***x***) of $sample_i$, and its desired output (***d***)
  - Calculate the net value (***v***),
  - Calculate actual output (***y***) using ***Linear*** activation function,
  - Calculate the ***error = d – y***,
  - Update the weights (***new weights = old weights + eta * error * x***), note: old weights is $\begin{bmatrix} b \\ W1i \\ W1j \end{bmatrix}$
- **Draw line:** line equation is ***W1i * Xi + W1j * Xj + b = 0***

Task 2 - Workflow

- **Testing Phase:**
  1. Given a sample x
  2. Calculate the net value (v),
  3. Calculate actual output (y) using ***signum*** activation function,
  4. **Output:** y (Class ID).

- **Evaluation:** build the confusion matrix and overall accuracy

Thank you