TRAINING SIMPLE ML ALGORITHMS FOR CLASSIFICATION

- ARTIFICIAL NEURONS:
  - IN the early days of ML McCulloch & pits described a Neuron as a simple logic gate.
  - They argued that neurons recieve some input, and based in this input, modify their output.
  - Based on this theory, Rosenblatt published the first perceptron
  - The perceptron was able to automatically learn weights & biases that lead the transmission of a signal.

FORMAL DEFINITION OF AN ARTIFICIAL NEURON

- The decision function $\sigma(z)$ takes a linear combination of input values, x, & their corresponding weights, w. (z = the net input ($z = w_1 x_1 + w_2 x_2 + \dots + w_m x_m$)

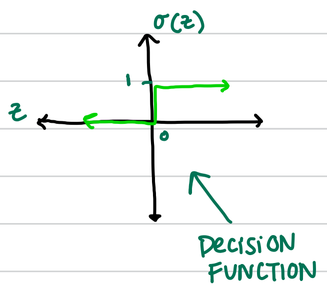
$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

- If x_i is greater than some threshold (θ), we predict class 1, otherwise 0.

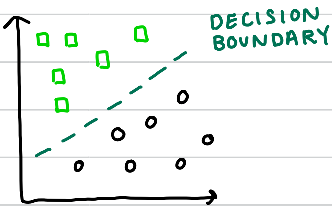
this is a unit step func. $\left[\quad \sigma(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{else} \end{cases} \right.$ $\dashrightarrow$ $z - \theta \geq 0$ can be described as our bias

$$\therefore \; z = w^T x + \underbrace{b}_{(-\theta)}$$

- IN this example, the decision function is the unit step function, which leads to a decision boundary that is linear.



→



THE PERCEPTRON LEARNING RATE:

ROSENBLATT'S rule IS AS FOLLOWS:

1. INITIALIZE the weights & bias units to zero or small random #s.
2. FOR EACH TRAINING EXAMPLE x^i
   a. compute $\hat{y}(i)$ predict
   b. update weights & biases

TRAINING SIMPLE ML ALGORITHMS FOR CLASSIFICATION

We can formally write the update of the bias unit & weights as:

where

$w_j := w_j + \Delta w_j$ $\quad \Delta w_j = \eta \, (y^i - \hat{y}^i) \, x_j^i$

&

$b := b + \Delta b$ $\quad \Delta b = \eta \, (y^i - \hat{y}^i)$

η = learning rate (between 0.0 & 1.0)
y^i = true class label
$\hat{y}^i$ = predicted class label

→ the bias unit & all weights are updated simultaneously

→ if the predictions are correct $\cancel{\Delta m}^{0}$ & $\cancel{\Delta b}^{0}$

eg.

$y_i = 1 \quad \hat{y}_i = 1 \quad \therefore \Delta w_j = \eta (\cancel{1-1})^{0} \cdot x_j^i \, , \Delta b = \eta (\cancel{1-1})^{0}$

→ if the predictions are incorrect:

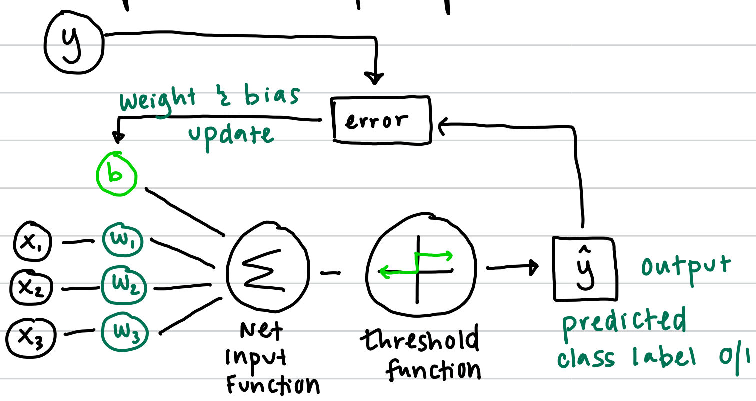
$y_i = 1 \quad \hat{y}_i = 0 \quad \therefore \Delta w_j = \eta (1-0) \cdot x_j^i = \eta \cdot x_j^i \;\; \& \;\; \Delta b = \eta (1-0) = \eta$

- with binary labels $(y^i - \hat{y}^i) \in \mathbb{R}[-1, 0, 1]$

✱ CONVERGENCE OF A PERCEPTRON CAN ONLY HAPPEN IF THE CLASSES ARE LINEARLY SEPERABLE

→ if the classes cant be seperated, we can set a maximum # of epochs (passes over the training dataset) or a threshold for the # of tolerated misclassification.

general concept of a perceptron:



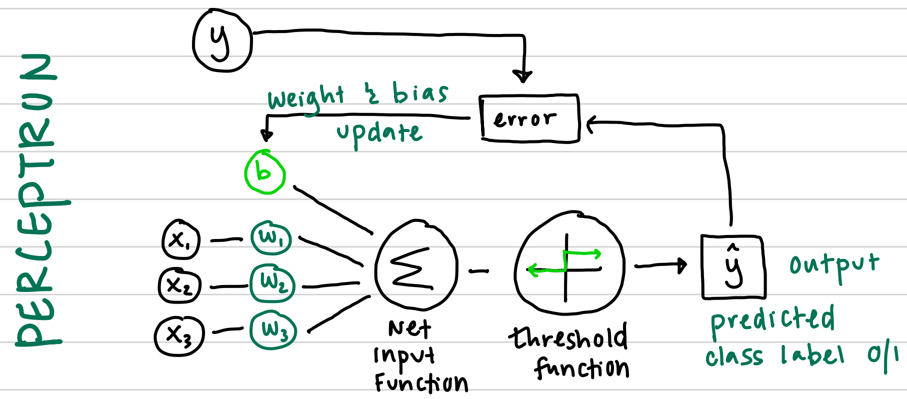
✱ SEE THE PERCEPTRON NOTEBOOK

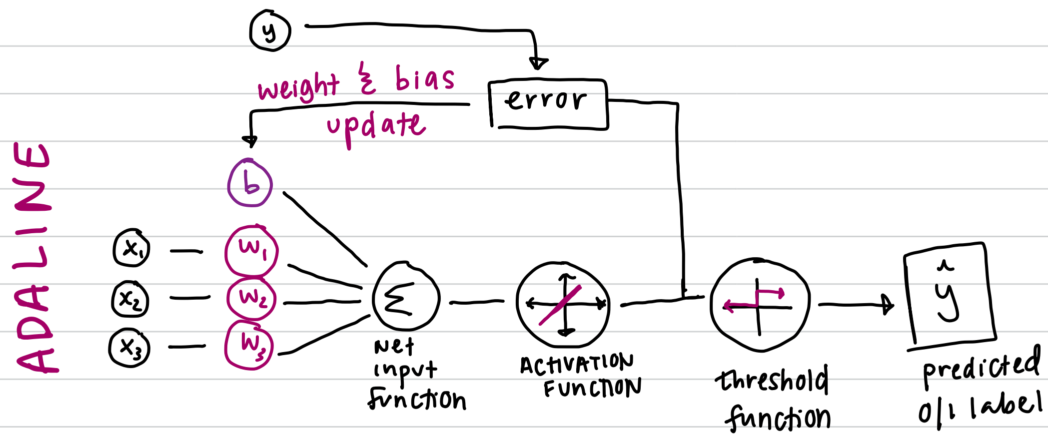
TRAINING SIMPLE ML ALGORITHMS FOR CLASSIFICATION

→ The next step more complex is an ADaptive LInear NEuron (ADALINE), where the weights/biases are updated based on a linear activation function, as opposed to a unit step function.

→ In adaline the linear activation function is simply the identity of the function $\sigma(z)=z$

→ the activation function is used to learn weights, but a threshold function is still used to make a final prediction.





MINIMIZING LOSS FUNCTION W/ GRADIENT DESCENT

- Objective function: loss or cost function we want to minimize
- In Adaline, the loss function (L) is the Mean Squared Error between predicted & true outcomes.

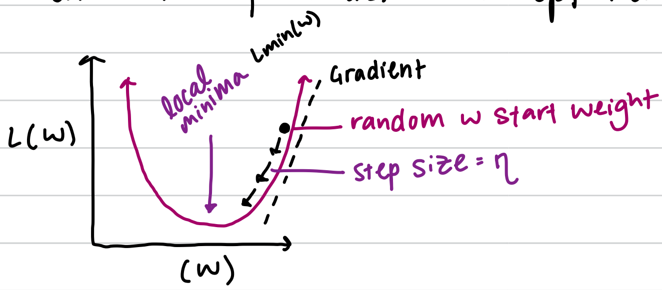
$$L(w,b) = MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y^i - \sigma(z^i)\right)^2$$

- The advantage of a linear activation function is that the function can be differentiated.

TRAINING SIMPLE ML ALGORITHMS FOR CLASSIFICATION

GRADIENT DESCENT CONT...

- gradient descent is good @ finding local minimas in our loss function, which will help us determine optimal w & bs.



- using gradient descent, we update the model parameters by taking a step in the opposite direction of the gradient, $\nabla L(w,b)$ of our loss function $L(w,b)$:

$$w := w + \Delta w \qquad \qquad \Delta w = -\eta \nabla_w L(w,b)$$

where:

$$b := b + \Delta b \qquad \qquad \Delta b = -\eta \nabla_b L(w,b)$$

- to compute the gradient of the loss function, we compute the partial derivative w/ respect to each weight

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

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

$$\Delta w_j = -\eta \cdot \frac{\partial L}{\partial w_j} \quad \& \quad \Delta b = -\eta \frac{\partial L}{\partial b}$$

$$w := w + \Delta w \qquad b := b + \Delta b$$

PDEs!

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

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

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

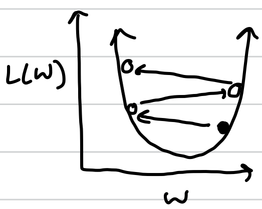
$$= \frac{2}{n} \sum_i \left(y^i - \sigma(z^i)\right) \left(\cancel{\frac{\partial y^i}{\partial w_j}}^{\,0} - \frac{\partial w_j x_j^i}{\partial w_j}^{\,1} + \cancel{\frac{\partial b}{\partial w_j}}^{\,0} \right)$$

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

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

- **THIS IS CLASSIFIED AS BATCH GRADIENT DESCENT BECAUSE THE WEIGHT UPDATE IS BASED ON ALL EXAMPLES IN THE DATASET**

If the learning rate is too large: we will move away from the minima.



TRAINING SIMPLE ML ALGORITHMS FOR CLASSIFICATION

FEATURE SCALING can help our gradient descent loss functions converge.

- STANDARDIZATION is a form of feature scaling in which the mean of each feature so that it is centered @ zero & each feature has a standard dev. of 1 (unit variance)

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

mean

standard deviation

- Helpful in gradient descent because weights to converge one unscaled feature on a vastly different scale than other features might destabilize other features.

★ See Adaline jupyter notebook.

STOCHASTIC GRADIENT DESCENT:

- Normal gradient descent is calculated from the whole training set, however, if our data set has million of points, this can be very computationally expensive.

- an alternative approach is stochastic gradient descent (SGD) where we update the weights & biases incrementally for each training example.

$$\Delta w_j := \eta\,(y^i - \sigma(z^i))\,x_j^i \qquad \Delta b = \eta\,(y^i - \sigma(z^i))$$

- It converges faster than GD, but is slightly more noisy which can help find global minima as opposed to local minima.

★ IMPORTANT TO SHUFFLE DATA BEFORE SGD

- ONLINE LEARNING is where our model is trained on the fly, such as where we are continuously adding data points. Also allows us to discard data after training.