Binary Logistic Regression

# Dataset preparation:

1. Use dataset [diabetes](https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv). Code for loading dataset into 2D python list: [here](https://colab.research.google.com/drive/1vzsOMsZHU4O5kLn_vUR5XxYrWv_Waimr?usp=sharing)
2. Randomly Split the dataset into Training (70%), Validation (15%) and Test (15%) set

# Train (update :

1. **for** each sample, ***X*** *= [x1, x2, … , xn]* **in** TRAINING set:
2. concatenate 1 and turn it into***X’*** *= [x1, x2, …, xn, 1]*
3. randomly initialize 𝚹 = [𝚹1, 𝚹2, …, 𝚹(n+1)] within 0 to 1

// 𝚹1, 𝚹2, …: weights, 𝚹(n+1): bias

1. *max\_iter* = 500, *lr* = 0.01
2. *history* = list()
3. **for** itr **in** [1, *max\_iter*]:
4. *TJ = 0* // total cost
5. **for** each sample, ***X’***, in TRAINING set:
6. *z = X’ . 𝚹* // use np.dot function
7. *h* = sigmoid(*z*) // sigmoid available in python
8. *J = - y log (h) - (1-y) log (1-h)* // *h* = pred label, *y* = true label
9. *TJ = TJ + J*
10. ***dv*** *= X’ . (h-y)* // dim(*dv*)=n+1
11. *𝚹 = 𝚹 - dv \* lr* // dim(*𝚹*)=n+1, *lr* = learning rate
12. *TJ = TJ / N\_train* // *N\_train* = #training samples
13. append *TJ* into *history* // average loss

# Validation:

1. *correct* = 0
2. **for** each sample ***V’*** in the **VALIDATION** set:
3. *z = V’.𝚹*
4. *h* = sigmoid(*z*)
5. **if** *h >= 0.5: h = 1*
6. **else**: *h = 0*
7. **if** *h == y: correct = correct + 1*
8. *val\_acc = correct \* 100 / N\_val* // *N\_val* = #validation samples

* Calculate validation accuracy (*val\_acc*) for *lr* = 0.1, 0.01, 0.001 and 0.0001 (*max\_iter* = 500)
* Make a table with 2 columns: learning rate *lr* and *val\_acc*
* Now, take the *lr* with maximum *val\_acc*
* Calculate *test accuracy* for *max\_iter* = 500 and the **chosen *lr*** in the previous step
* Plot the train\_loss (history) vs epoch (iteration) graph

# Instruction

* Submit a .ipynb file and a report ([report template](https://docs.google.com/document/d/1oeW3HAkndiidFKLQZvCp-mQTH7xkbJJb-sBBMs8KHYM/edit?usp=sharing)) .pdf file.
* **DO NOT USE LIBRARIES SUCH AS: "Sklearn", "Scikit learning" or "pandas" for this assignment**
* **Copying will result in -100% penalty**

# Marks Distribution

(1) Dataset loading, train-val-test split:2

(2) Training code: 8

(3) Validation/ test code: 5

(4) l.r. and val\_acc table: 2.5

(5) train\_loss vs epoch graph plot for the best l.r.: 2.5

**Task (2)-(5) have to be done without using sklearn like libraries.**

**Your marks will fully depend on your viva and understanding.**

# 

# Resources

[Logistic\_Regression\_CRR.pdf](https://drive.google.com/file/d/1PyXb8KnJLNhWAJDW0IlAJSvm6AqmPyaI/view?usp=sharing)

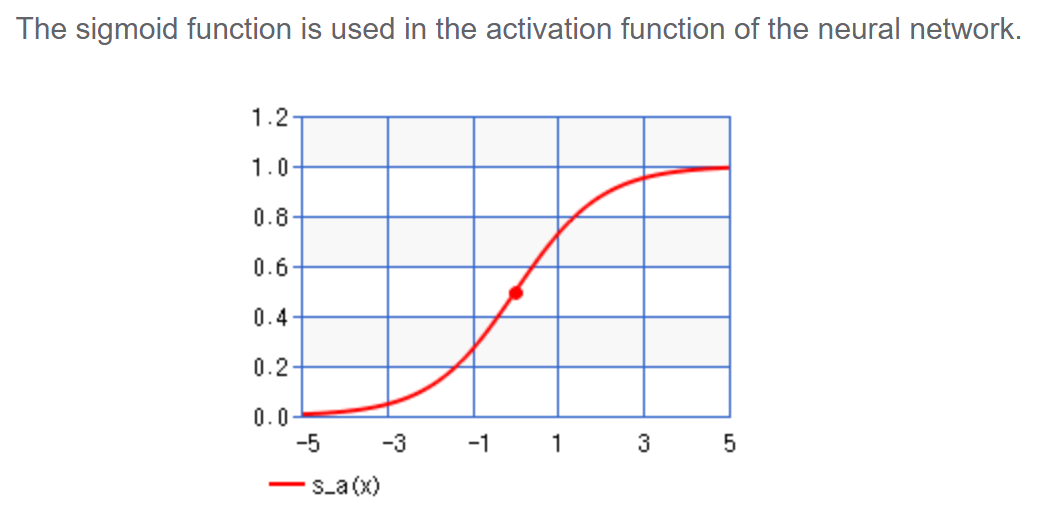
Labels = 0 or 1 ⇒ binary classification

**How to predict?**

Let, sample 1 of dataset, X\_1 = [x1, x2, x3, 1]

Weights, = [1, 2, 3, **4**] **4 is called bias**

Model/Prediction equation: ***z = X. = x11 + x2.3 + x33 + 4***. We update weights so that z can correctly predict the label of X\_1, but its value can be very big (>1) or very small (<0).

Solution: use activation function sigmoid

**sigmoid(z) =**

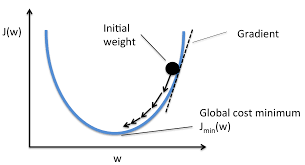
So, *h = sigmoid(z) is the predicted label of X1*

**How to update weights?**

**Gradient descent optimization**

Log loss function: J() = - y log(h) - (1-y) log(1-h) , y is the true label and h is the predicted label

The closer h is to y, the lesser the loss.



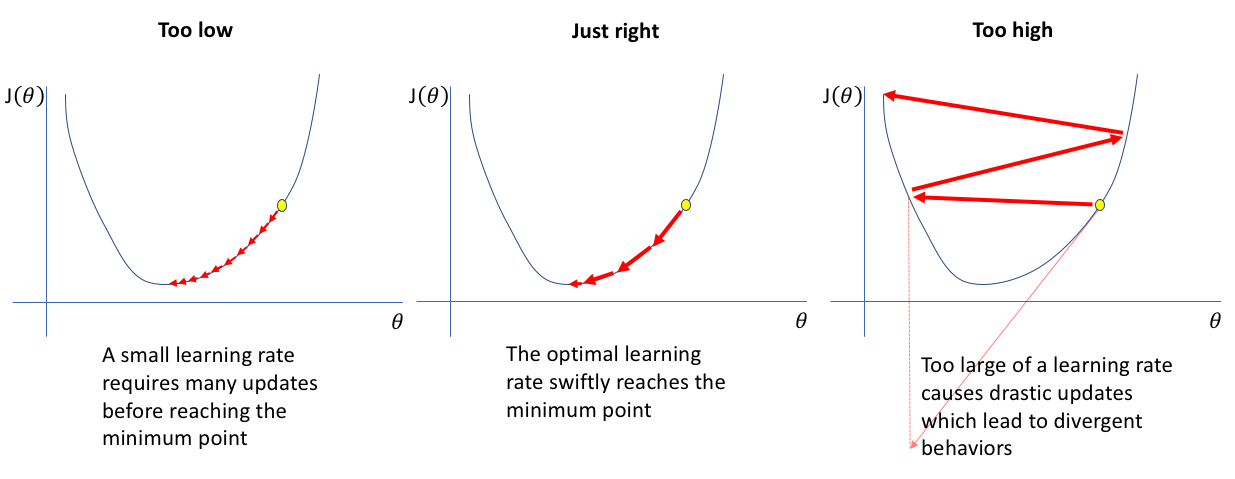
dv = Derivative of J() = **Gradient** = X(h-y)

If gradient +ve, we should decrease weights, else if gradient -ve, we should increase weights. So, update = - dv

However, weights may oscillate without reaching our desired value.

Solution: introduce learning rate lr **(0<lr<1) e.g. 0.01, 0.001, 0.0001**

= - dv \* lr



Weights are updated using the training set.

**How to choose the value of lr?**

Hyperparameter tuning using validation set.