Logistic Regression and Neural Networks

Binary classification

Logistic regression is an algorithm for **binary classification**

- **Input**: a vector $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ features vector (of the objects)

- **Output (target)** $y \in \{0, 1\}$

  1 – positive (in the class)
  0 – negative (not in the class)

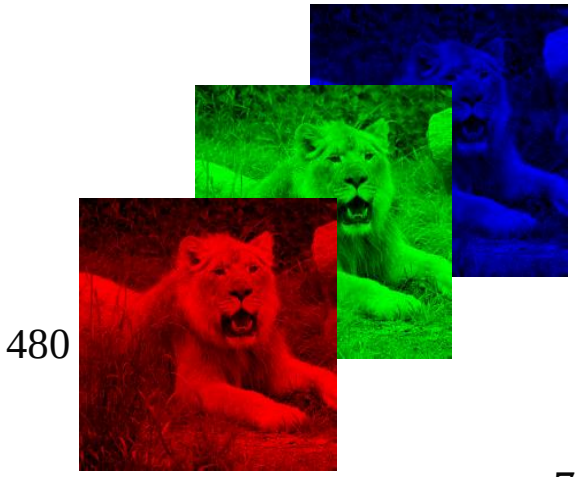
Example 1

Input: image

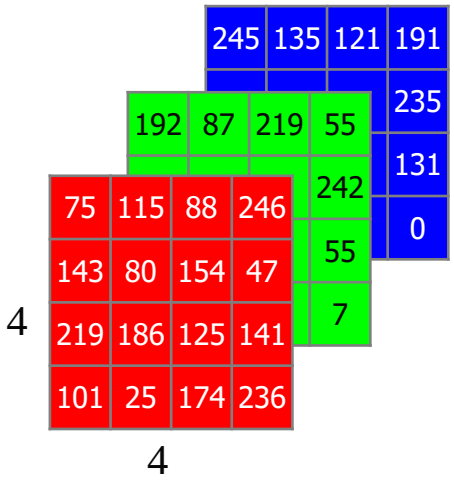
Output: classify (recognize) the image as: lion / not lion

y = 1 / 0

Which is the input vector?
Computer representation of an image:



Assuming a much smaller image (4 x 4 x 3)



Turn these pixel intensity values into a feature vector:
- unroll all pixel values into an input feature vector x (48 x 1)

$$x = \begin{bmatrix} 75 \\ 192 \\ 245 \\ 115 \\ \\ 141 \\ 55 \\ 131 \\ \vdots \\ 236 \\ 7 \\ 0 \end{bmatrix}$$

$n = 4 \times 4 \times 3 = 48$

For the initial image, the dimension of feature vector will be: n = 480 x 480 x 3 **= 691 200 !**

Example 2

Classify if a bank customer is going to churn ($y = 1$) or not ($y = 0$)

Dataset (bank customer churn):

$n = 9$ features

$m = 10\,000$ examples (samples)

Feature vector									Target
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y
Gender	Age	Credit Score	Tenure	Balance	Products Number	Credit Card	Active Member	Estimated Salary	Exited
0	45	619	2	0	1	1	1	2,846	1
0	44	608	1	77,941.31	1	0	1	3,035	0
0	45	502	8	148,484.54	3	1	0	3,058	1
0	42	699	1	0	2	0	0	2,719	0
0	46	850	2	116,725.06	1	1	1	2,470	0
1	47	645	8	105,792.88	2	1	0	3,663	1
1	53	822	7	0	2	1	1	1,306	0
0	32	376	4	106,993.47	4	1	0	3,150	1
1	47	501	4	132,107.50	2	0	1	2,401	0
1	30	684	2	125,181.61	1	1	1	2,346	0
1	34	528	6	94,875.55	2	0	0	2,489	0
1	27	497	3	0	2	1	0	2,425	0
0	37	476	10	0	2	1	0	1,579	0
0	28	549	5	0	2	0	0	4,356	0
0	38	635	7	0	2	1	1	2,249	0
1	48	616	3	133,110.35	2	0	1	2,221	0
1	56	653	1	123,320.68	1	1	0	1,222	1
0	27	549	9	0	2	1	1	1,379	0
1	48	587	6	0	1	0	0	3,813	0

Build a prediction model

Logistic regression

Logistic regression is an algorithm for **binary classification**

Given an input feature vector $x \in R^n$ **we want** an algorithm that can output a binary prediction:

$$y \in \{0, 1\}$$

In fact, **we better want** our model to predict the **chance (probability)** for the object described by a set of features x to be in the class

$$\hat{y} \in [0, 1] \qquad 0 \leq \hat{y} \leq 1$$

For the bank customer churn dataset, we should rather prefer a model that will attach a probability to the churn to make it easier for customer service to target low hanging fruits in their efforts to prevent churn.

Linear regression revisited

$$\hat{y} = ax + b$$

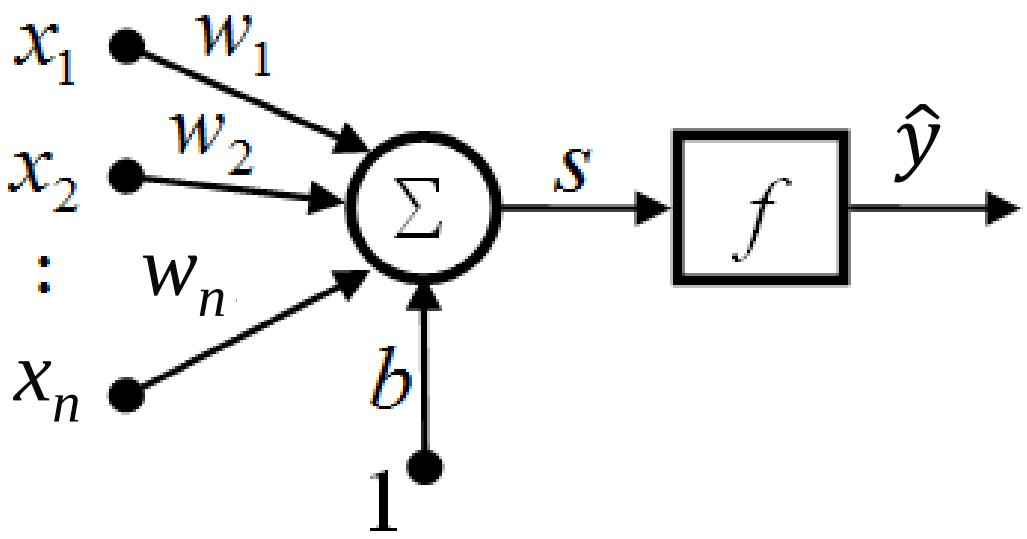
Simple linear regression

$$\hat{y} = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + b$$

Multiple linear regression

Neuron Model as Logistic Regression

Binary classification



$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

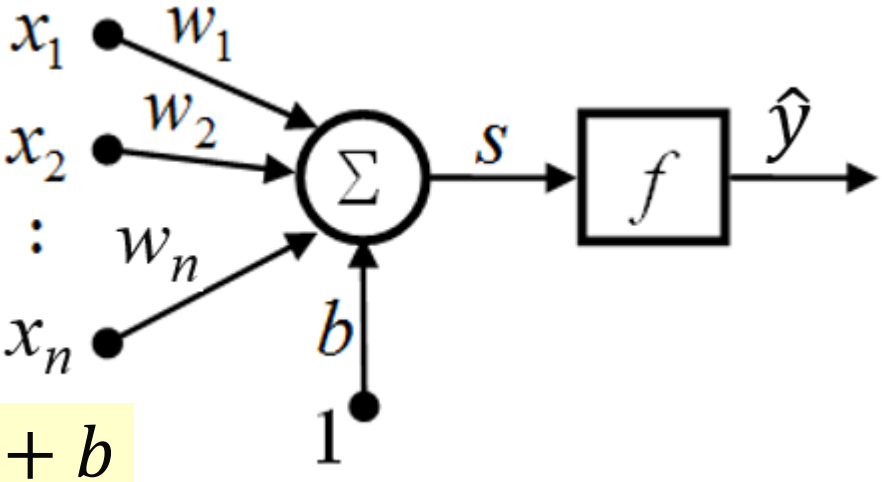
$\hat{y} \in [0, 1]$ $0 \leq \hat{y} \leq 1$

Σ - **summing component**

f - **activation function**

Neuron Model as Logistic Regression

Binary classification



$$s = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

The weighted sum s is in fact a **multiple linear regression**

If $n = 1$, the weighted sum s became a **simple linear regression**

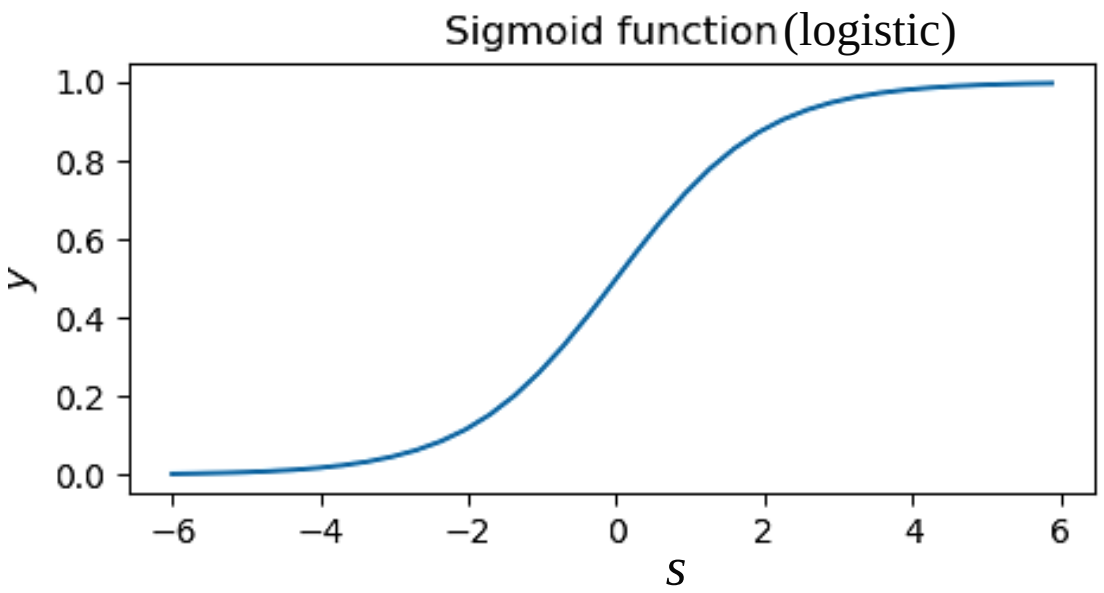
$$s = w_1 x_1 + b$$

Activation function f

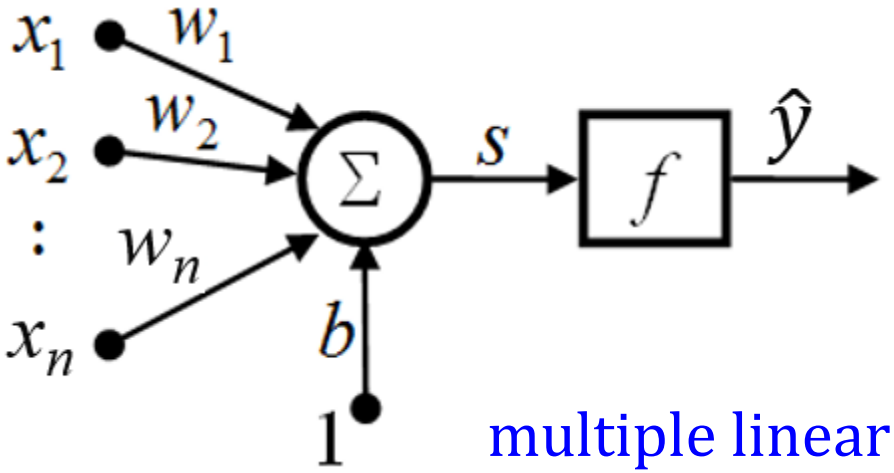
Sigmoid-type function

logistic function => *logsig*

$$\hat{y} = f(s) = \frac{1}{1+e^{-s}}$$



Neuron Model as Logistic Regression



Input vector
column vector

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

multiple linear regression

Weight vector
row vector

$$w = [w_1\ w_2\ \ldots w_n]$$

$$\hat{y} = f(s) = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

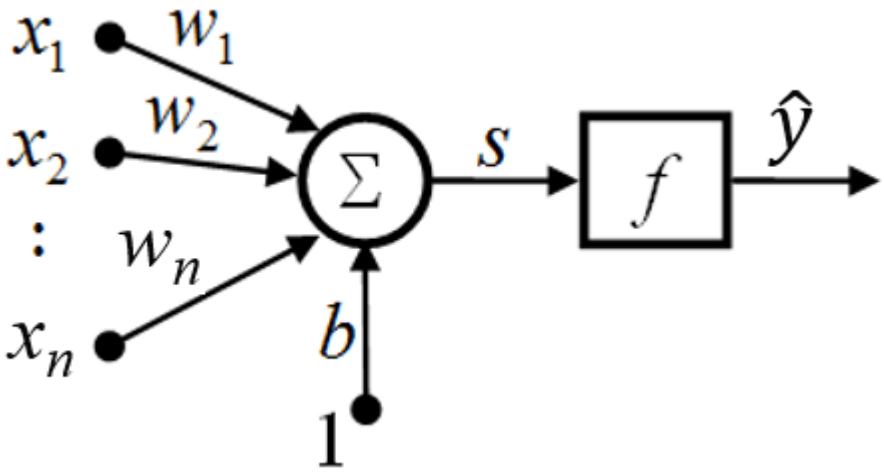
Bias b

$$s = \sum_{i=1}^{n} w_i x_i + b = wx + b$$

$$s = wx + b \quad \hat{y} = f(s)$$

$$\hat{y} = f(wx + b)$$

Problem

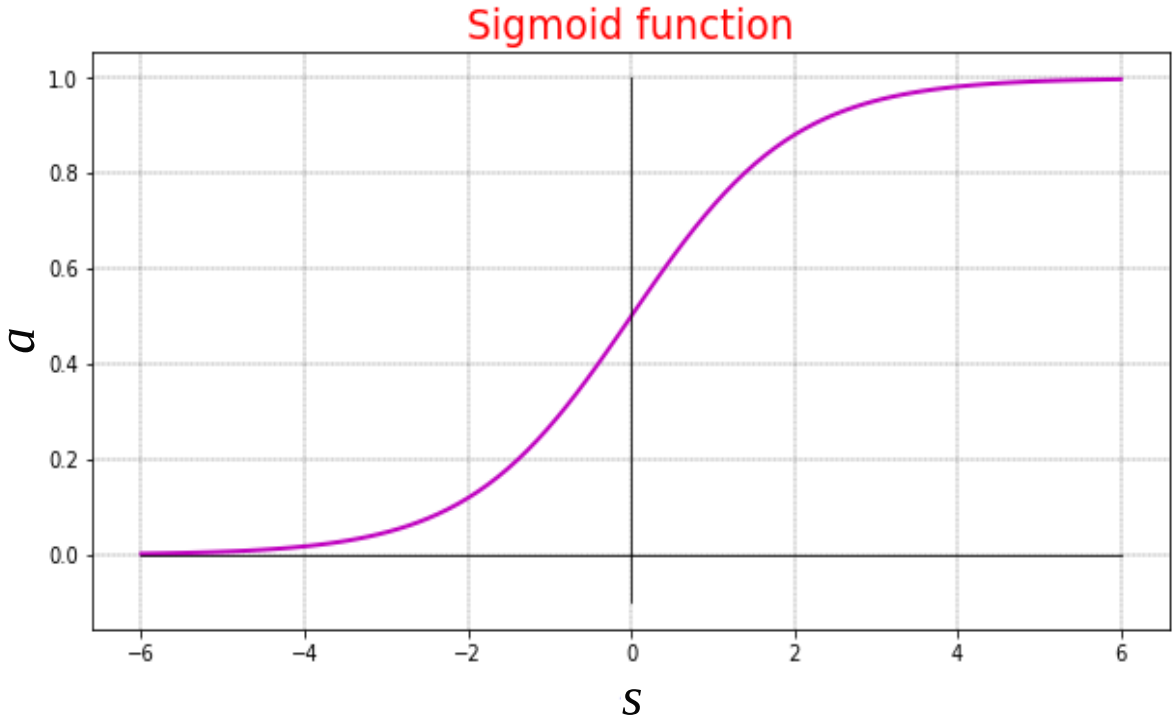


$n = 2$

$w = [\,0.25 \quad -0.5]$

$b = 2.5$

$x^{(1)} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$; $x^{(2)} = \begin{bmatrix} 4 \\ 15 \end{bmatrix}$;



$s = ?;\ \hat{y} = ?$

Implement the above neuron in Python

Phyton 3.5 (Spyder) implementation using NumPy module

```
23 #---------------------------------------
24 import numpy as np  # import the NumPy module
25 # %%
26 x = np.array([2, 4, -0.8])  # create the array for the input (vector)
27 w = np.array([0.25, -0.5, 1])  # create the array for the weight vector
28 b = -.99 # the bias
29
30 # %% compute s
31 # version 1
32 mx = w*x  # elementwise product
33 s1 = mx.sum()  # adding the elements of the array
34 # version 2
35 s2 = w.dot(x)  # using the matrix product
36 # %%
37 s=s2+b    # add bias
38 # apply activation function - Hyperbolic tangent
39 import math
40 y=math.tanh(s)
41 # printing section
42 print ('x=')
43 for item in x: print ('{:6.2f}'.format(item))  # format floating point numbers
44 # at least 6 characters (including sign and decimal point)
45 # with 2 after the decimal point
46 print ('\nw =',w)
47 print ('\nb = ',b)
48 print('\ns={:6.4f}'.format(s))
49 print('\ny = {:7.4f}'.format(y))
```

Phyton 3.5 (Spyder) results

```
x=
    2
    4
w = [0.25, -0.5]
b =  2.5

s=  1.0
y =  0.761594155955764 9
```

```
x=
    2
    4
    -0.3
w = [0.25, -0.5, 1]
b =  0.5

s= -1.3
y =  -0.861723159313306 3
```

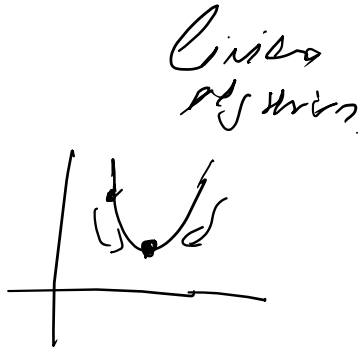
```
x=
    2
    4
    -0.8
w = [0.25, -0.5, 1]
b =  -0.99

s=-3.29
y =  -0.997228148322723 9
```

The cross-entropy cost function for logistic regression

$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ we want to predict $\hat{y}^{(i)} \approx y^{(i)}$

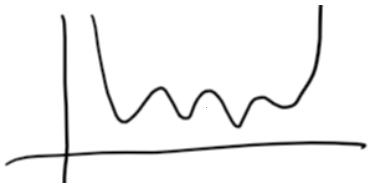
$$\hat{y}^{(i)} = f(wx^{(i)} + b) \qquad f(s^{(i)}) = \frac{1}{1+e^{-s^{(i)}}}$$



$x^{(i)}$ denotes the i^{th} example, $i = 1, 2, \ldots, m$

Quadratic loss (error) function:

$$L(\hat{y}, y) = (\hat{y} - y)^2$$



Using **quadratic** loss function, the optimization became noncovex; it can be stacked easily in local optima, slow convergence in the optimization – **not recommended** for logistic regression

The cross-entropy cost function for logistic regression

$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ we want to predict $\hat{y}^{(i)} \approx y^{(i)}$

The **cross-entropy loss function** is recommended for logistic regression:

$$\mathcal{L}(\hat{y}, y) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$$

$\mathcal{L}(\hat{y}, y)$ is non-negative

- all the individual terms are negative, since both logarithms are of numbers in the range [0, 1]
- there is a minus sign out the front of the sum.

If $y=1$; $\mathcal{L}(y, \hat{y}) = -\log \hat{y}$; $\log \hat{y}$ large: $\hat{y}$ large ($\to 1$)

$y=0$; $\mathcal{L}(y, \hat{y}) = -\log(1-\hat{y})$; $\log(1-\hat{y})$ large: $\hat{y}$ small ($\to 0$)

$\log 1 = 0$

The contribution to the cost will be low provided the actual output is close to the desired output.

Summing up, the cross-entropy is positive, and tends toward zero as the neuron gets better at computing the desired output, for all training inputs.

What is Entropy?

Entropy is a measure of the randomness, or we can say uncertainty in the probability distribution of some specific target.

The entropy of gases is higher than that of the solids because of the faster movement of particles in the latter which leads to a higher degree of randomness.

What is the difference between Entropy and Cross-Entropy?

Entropy just gives us the measure of the randomness in a particular probability distribution, but the requirement is somewhat different in machine learning which is to compare the difference between the probability distribution of the predicted probabilities and the true probabilities of the target variable.

And for this case, the cross-entropy function comes in handy and serves the purpose.

Due to this only we try to minimize the difference between the predicted and the actual probability distribution of the target variable. And as the value of the cost function decreases the performance of the Machine Learning model improves.

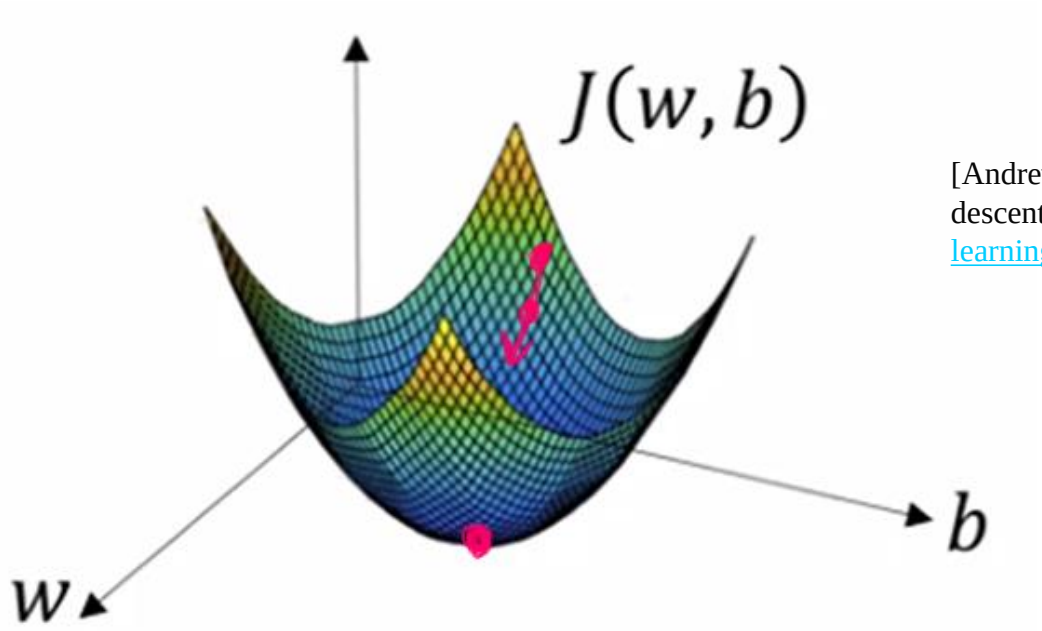
[https://www.geeksforgeeks.org/cross-entropy-cost-functions-used-in-classification/]

The cross-entropy cost function for logistic regression

Cost function (across m training examples)

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}) \right]$$

Find w and b that minimize $J(w, b)$

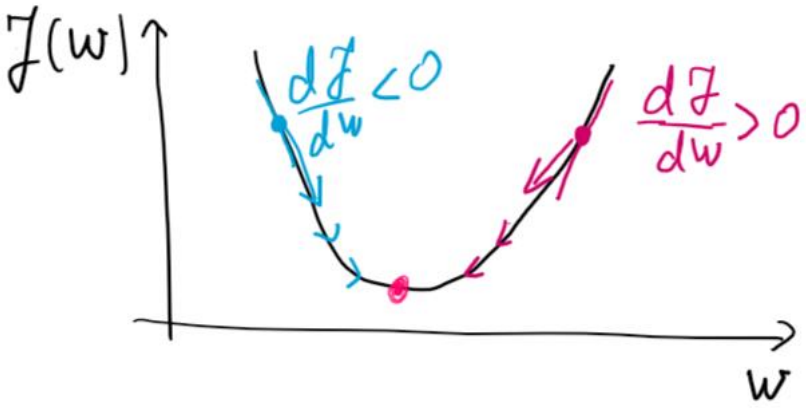


[Andrew Ng, Logistic regression as a neural network. Gradient descent, https://www.coursera.org/learn/neural-networks-deep-learning/lecture/A0tBd/gradient-descent]

Gradient descent

Cost function (across m training examples)

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)}) \right]$$



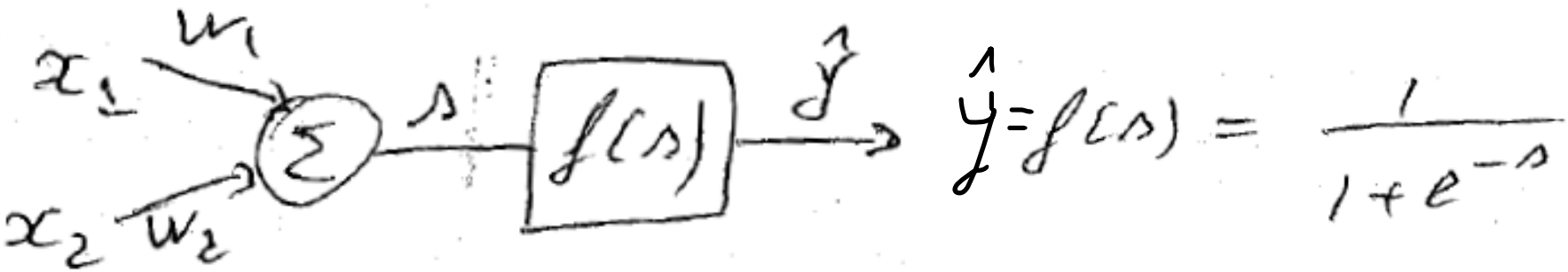
$$w := w - \eta \frac{dJ(w,b)}{dw}$$

Update the weights and biases, in each training epoch, according to gradient descent, to reach the minimum of cost function.

$$\begin{cases} w := w - \eta \dfrac{dJ(w,b)}{dw} \\ b := b - \eta \dfrac{dJ(w,b)}{db} \end{cases}$$

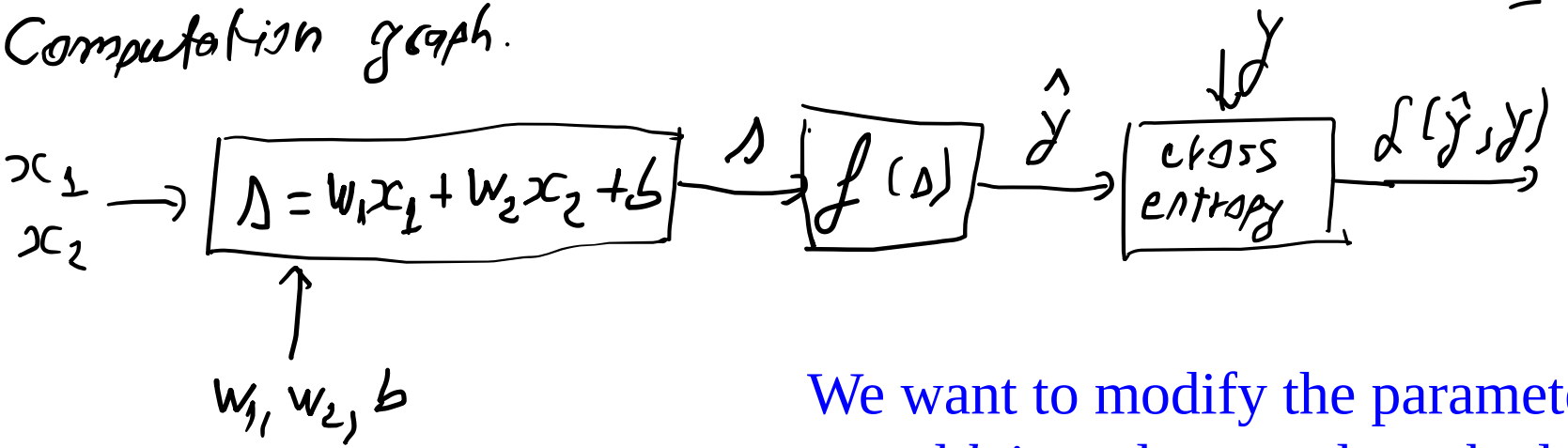
Logistic Regression. Gradient Descent. Back Propagation

Consider the case for only 2 features



$$\mathcal{L}(\hat{y}, y) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$$

Computation graph.



We want to modify the parameters w and b in order to reduce the loss

How does $\mathcal{L}$ depends on model parameters ?

$$\frac{\partial \mathcal{L}}{\partial w_1} \; ; \; \frac{\partial \mathcal{L}}{\partial w_2} \; ; \; \frac{\partial \mathcal{L}}{\partial b} \; ?$$

dW db

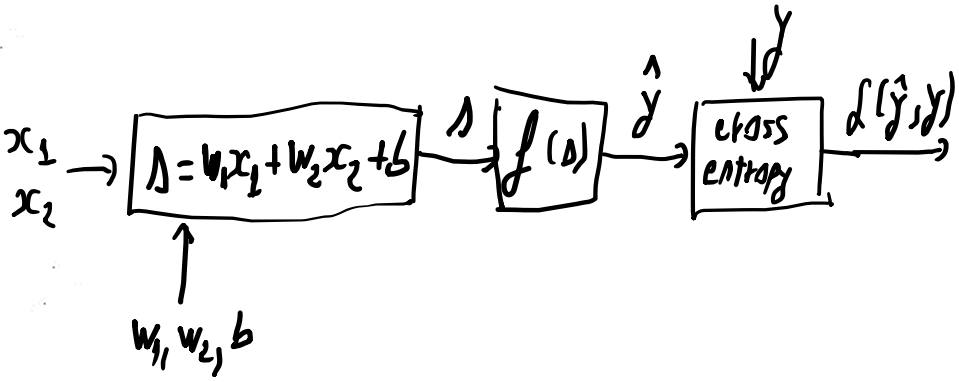
$$w_1 := w_1 - \eta \frac{\partial \mathcal{L}}{\partial w_1}$$

$$w_2 := w_2 - \eta \frac{\partial \mathcal{L}}{\partial w_2}$$

$$b := b - \eta \frac{\partial \mathcal{L}}{\partial b}$$

η learning rate

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w_1} = \dfrac{\partial \mathcal{L}}{\partial s} \cdot \dfrac{\partial s}{\partial w_1} = \dfrac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \dfrac{\partial \hat{y}}{\partial s} \cdot \dfrac{\partial s}{\partial w_1} \\ \dfrac{\partial \mathcal{L}}{\partial w_2} = \dfrac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \dfrac{\partial \hat{y}}{\partial s} \cdot \dfrac{\partial s}{\partial w_2} \\ \dfrac{\partial \mathcal{L}}{\partial b} = \dfrac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \dfrac{\partial \hat{y}}{\partial s} \cdot \dfrac{\partial s}{\partial b} \end{cases}$$



$$\begin{cases} \dfrac{\partial s}{\partial w_1} = \dfrac{\partial (w_1x_1 + w_2x_2 + b)}{\partial w_1} = x_1 \\ \dfrac{\partial s}{\partial w_2} = x_2 \\ \dfrac{\partial s}{\partial b} = 1 \end{cases}$$

$$\frac{d\hat{y}}{ds} = \frac{d\left(\frac{1}{1+e^{-s}}\right)}{ds}$$

$$\left(\frac{1}{g}\right)' = -\frac{g'}{g^2}$$

$$(e^g)' = e^g \cdot g'$$

$$\frac{d\hat{y}}{ds} = -\frac{e^{-s}\cdot(-1)}{(1+e^{-s})^2} = \frac{e^{-s}}{(1+e^{-s})^2} = \frac{1}{1+e^{-s}} \cdot \frac{e^{-s}}{1+e^{-s}}$$

$$\frac{d\hat{y}}{ds} = \underbrace{\frac{1}{1+e^{-s}}}_{\hat{y}} \cdot \underbrace{\left(1 - \frac{1}{1+e^{-s}}\right)}_{\hat{y}} = \hat{y}(1-\hat{y})$$

$$\boxed{\frac{d\hat{y}}{ds} = \hat{y}(1-\hat{y})}$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\partial\left(-\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right]\right)}{\partial \hat{y}}$$

$$(\log g)' = \frac{1}{g} \cdot g'$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\left[\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \cdot (-1)\right] = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}}$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial s} = \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \cdot \hat{y}(1-\hat{y})$$

$$\frac{d\hat{y}}{ds} = \hat{y}(1-\hat{y})$$

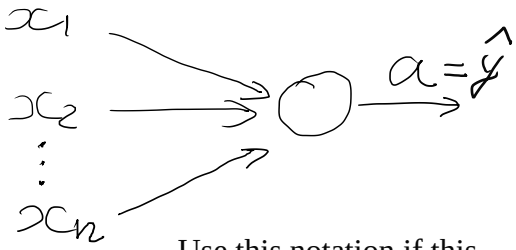
$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{d\hat{y}}{ds} = \frac{-y + y\hat{y} + \hat{y} - y\hat{y}}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y})$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial s} = \hat{y} - y.$$

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w_1} = (\hat{y} - y) x_1 \\ \dfrac{\partial \mathcal{L}}{\partial w_2} = (\hat{y} - y) x_2 \\ \vdots \\ \dfrac{\partial \mathcal{L}}{\partial b} = \hat{y} - y \end{cases} \qquad \begin{cases} w_1 := w_1 - \eta (\hat{y} - y) x_1 \\ w_2 := w_2 - \eta (\hat{y} - y) x_2 \\ \vdots \\ b := b - \eta (\hat{y} - y) \end{cases}$$

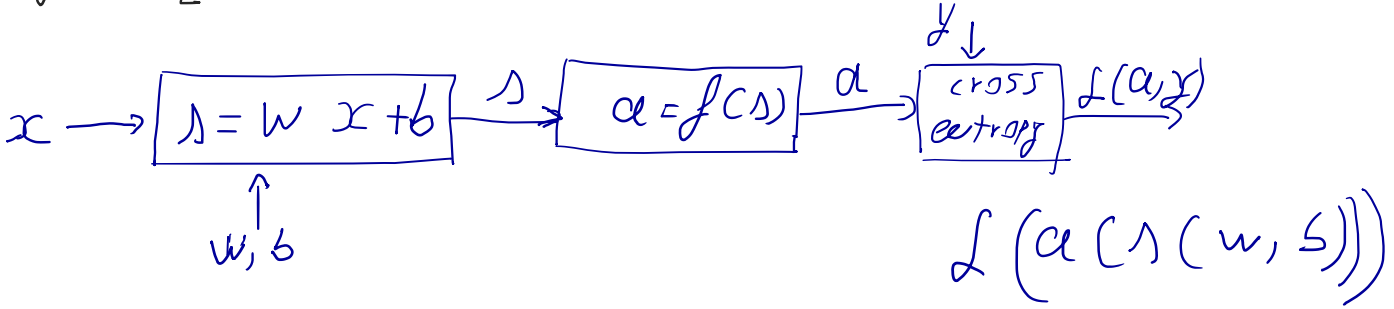
Logistic regression – forward propagation, gradients computing, parameters updating



Use this notation if this neuron is not the output one, but an intermediate one (in an intermediate layer)

- Sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$
- Loss function $\mathcal{L}(a, y) = -\left[y \log a + (1-y) \log (1-a)\right]$

Forward propagation



Computing the gradients

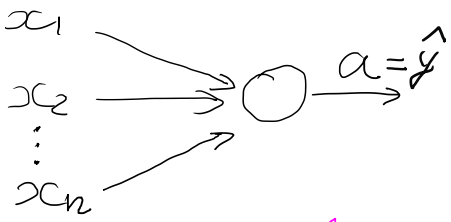
How does the loss function depend on training parameters w and b?

$$\frac{d\mathcal{L}(a,y)}{dw} = \frac{d\mathcal{L}(a,y)}{dz} \cdot \frac{dz}{dw} = \frac{d\mathcal{L}(a,y)}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{dw} = dz \cdot \frac{dz}{dw}$$

$$\frac{d\mathcal{L}}{db} = \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{db} = dz \cdot \frac{dz}{db}$$

Logistic regression – forward propagation, gradients computing, parameters updating

- Sigmoid function $\sigma(s) = \frac{1}{1+e^{-s}}$
- Loss function $\mathcal{L}(a, y) = -\left[y \log a + (1-y) \log (1-a)\right]$



$$da = \frac{d\mathcal{L}(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{a}$$

$$ds = da \cdot \frac{da}{ds} = \left(-\frac{y}{a} + \frac{1-y}{a}\right) \cdot a(1-a) = a - y$$

$$\frac{ds}{dw} = x \qquad \frac{ds}{db} = 1$$

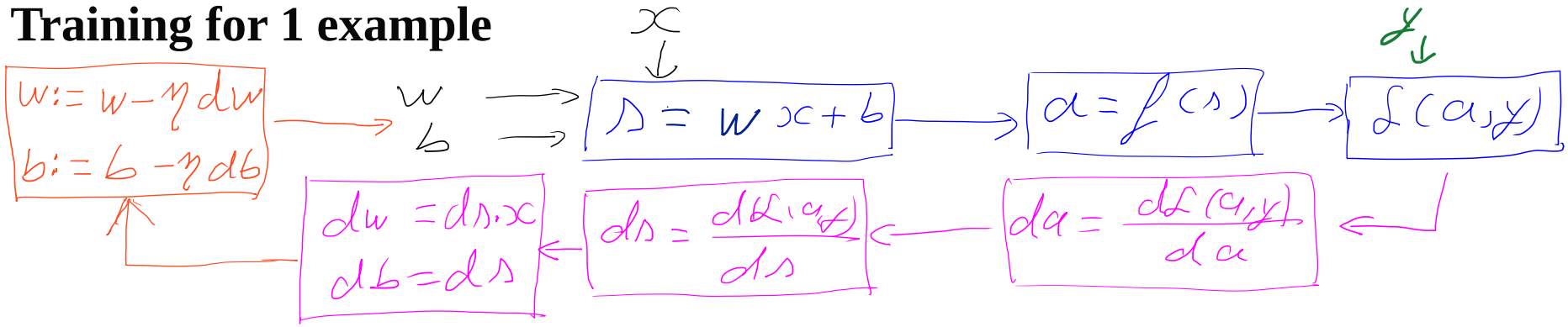
$$\begin{cases} dw = ds \cdot x = (a-y)x \\ db = ds \cdot 1 = a - y \end{cases}$$

Parameters updating

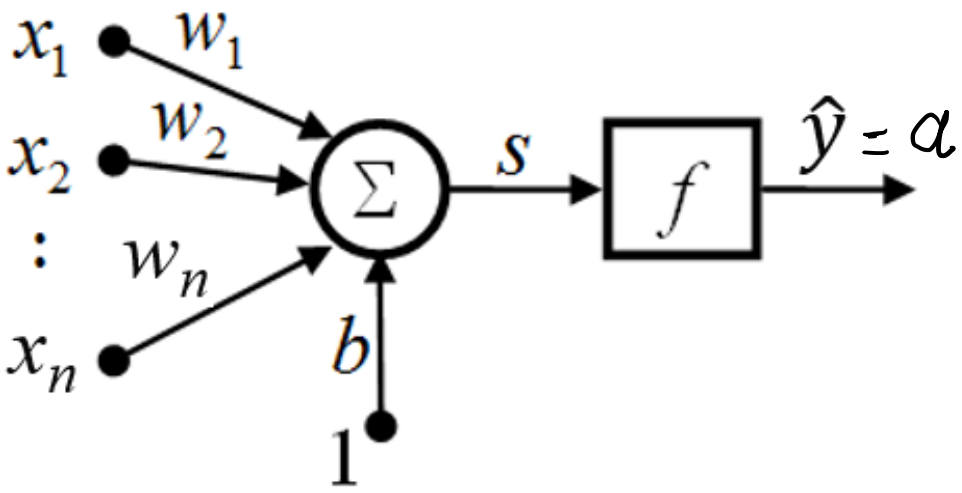
$$w := w - \eta\, dw$$
$$b := b - \eta\, db$$

η – learning rate

Training for 1 example



Logistic Regression. Gradient Descent. Back propagation on m examples



$\hat{y} = a$

Use this notation if this neuron is not the output one, but an intermediate one (in an intermediate layer)

$$z = wx + b$$

$$\hat{y} = a = f(z) = f(wx + b)$$

$$L(a, y) = -[y \log a + (1 - y) \log (1 - a)]$$

$$\frac{\partial L}{\partial w_1} = (a-y)x_1 \qquad w_1 := w_1 - \eta(a-y)x_1$$

$$\frac{\partial L}{\partial w_2} = (a-y)x_2 \qquad w_2 := w_2 - \eta(a-y)x_2$$

$$\vdots \qquad\qquad \vdots$$

$$\frac{\partial L}{\partial w_n} = (a-y)x_n \qquad w_n := w_n - \eta(a-y)x_n$$

$$\frac{\partial L}{\partial b} = a-y \qquad b := b - \eta(a-y)$$

For *m* examples: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Cost function $J(a, y) = J(w, b)$

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

$$a^{(i)} = f(w x^{(i)} + b)$$

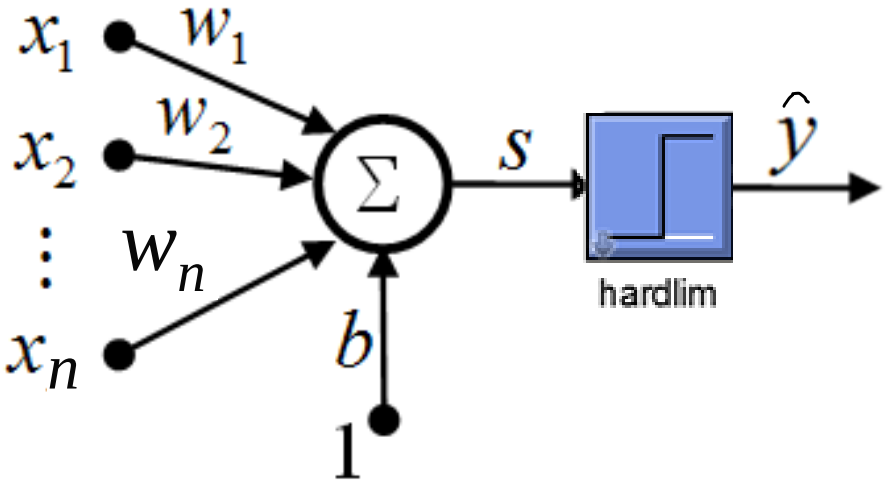
$$\begin{cases} \dfrac{\partial J(w,b)}{\partial w_1} = \dfrac{1}{m} \displaystyle\sum_{i=1}^{m} \dfrac{\partial \mathcal{L}(a^{(i)}, y^{(i)})}{\partial w_1} = \dfrac{1}{m} \displaystyle\sum_{i=1}^{m} (a^{(i)} - y^{(i)}) x_1^{(i)} \\ \vdots \\ \dfrac{\partial J(w,b)}{\partial b} = \dfrac{1}{m} \displaystyle\sum_{i=1}^{m} \dfrac{\partial \mathcal{L}(a^{(i)}, y^{(i)})}{\partial b} = \dfrac{1}{m} \displaystyle\sum_{i=1}^{m} (a^{(i)} - y^{(i)}) \end{cases}$$

For *m* examples: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Parameters updating

$$\begin{cases} w_1 := w_1 - \eta \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)}) x_1^{(i)} \\ w_2 := w_2 - \eta \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)}) x_2^{(i)} \\ \vdots \qquad\qquad \vdots \\ w_n := w_n - \eta \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)}) x_n^{(i)} \\ b := b - \eta \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)}) \end{cases}$$

Perceptron



Input vector $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

Weight vector $w = [w_1\ w_2\ \ldots w_n]$

$$s = wx + b \quad \hat{y} = f(s)$$

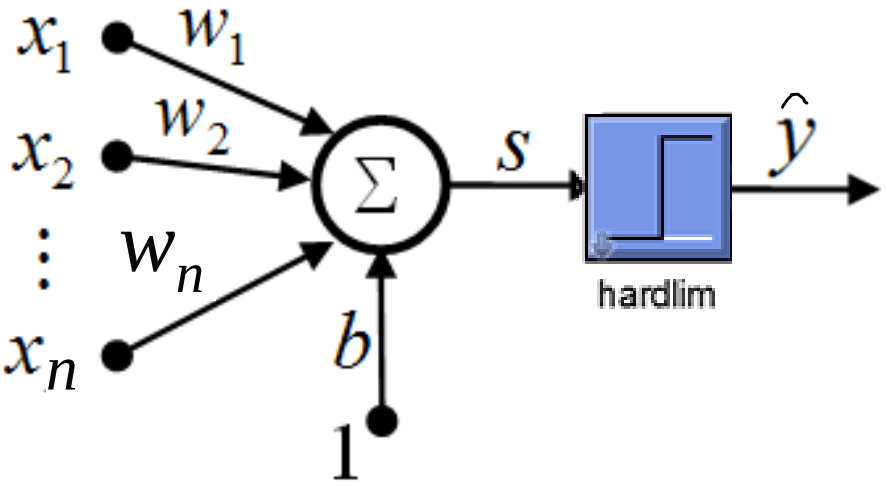
$$\hat{y} = f(wx + b)$$

$$\hat{y}(s) = \begin{cases} 0, & s < 0 \\ 1, & s \geq 0 \end{cases}$$

A **Perceptron** is a **binary classifier** that maps its input *x* (a real-valued vector) to an output value *y* (*y* single binary value, 0 or 1)

- Rosenblatt [Rose61] created many variations of the perceptron.
- One of the simplest: single-layer network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector.
- The **training technique** is called the **perceptron learning rule**.
- The perceptron generated great interest due to its ability to generalize from its training vectors and learn from initially randomly distributed connections.
- The perceptron is especially suited for **simple problems** in pattern classification.
- They are fast and reliable networks for the problems they can solve.

Perceptron Learning rule



Input vector Weight vector

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad w = [w_1\ w_2\ \dots w_n]$$

$$\hat{y} - y = \begin{cases} 0 & \text{if } \hat{y} = y = 0 \text{ or } \hat{y} = y = 1 \\ 1 & \text{if } \hat{y} = 1;\ y = 0 \\ -1 & \text{if } \hat{y} = 0;\ y = 1 \end{cases}$$

$$\begin{cases} w_1 := w_1 - (\hat{y} - y)x_1 \\ w_2 := w_2 - (\hat{y} - y)x_2 \\ \vdots \\ b := b - (\hat{y} - y) \end{cases}$$

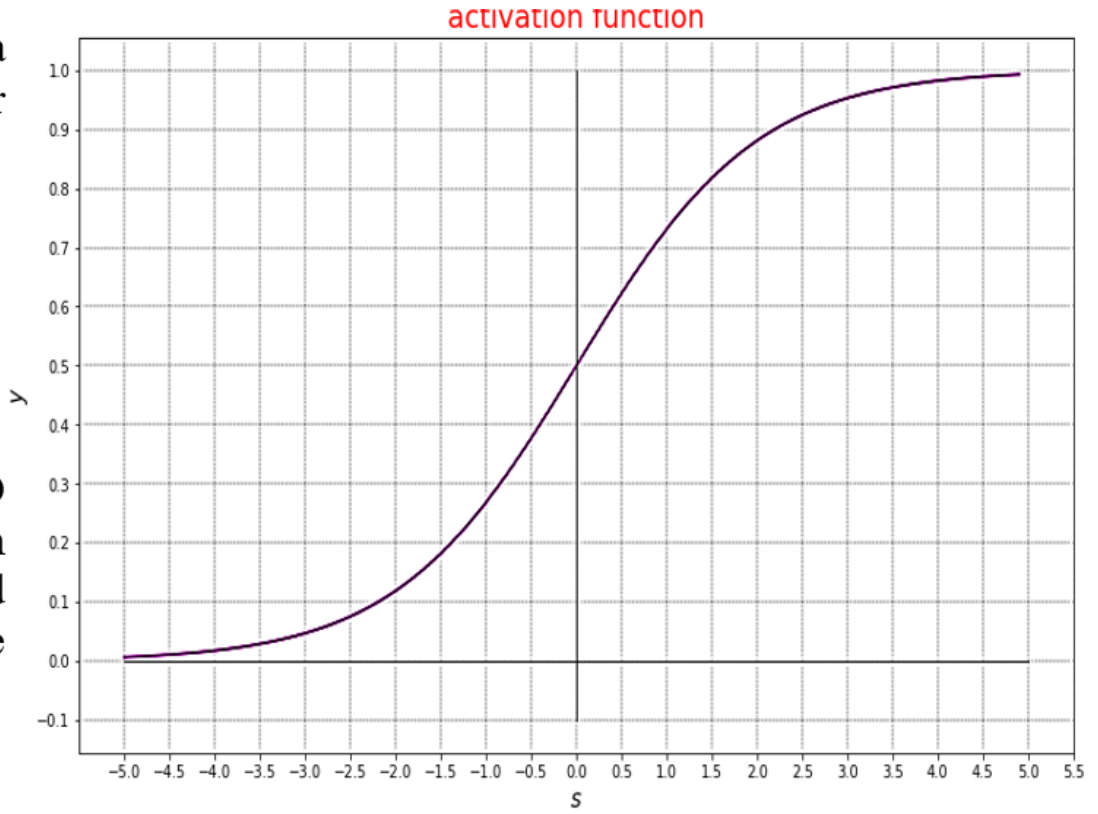
P1. 2p The training dataset to train a logistic regression model (one neuron) for two inputs and one output is:

$$x^{(1)} = \begin{bmatrix} -2 \\ +4 \end{bmatrix}, \quad y^{(1)} = 0$$

$$x^{(2)} = \begin{bmatrix} 3 \\ -8 \end{bmatrix}, \quad y^{(2)} = 1$$

The activation function is given in the next image. To fit the model, we are using BGD (batch gradient descent) optimization algorithm with a learning rate of 0.08, and the cross-entropy loss / cost function. The initial values of training parameters are:

w=[-0.5 0.4] b = [0]



a) **0.5p** For the initial parameters, determine the predictions and the loss functions for each training example, using forward propagation. Determine the cost function.

b) **0.5p** Using backward propagation, complete one training epoch and determine the updated values for all training parameters.

c) **0.5p** Determine the predictions for the training data set, after the 1st training epoch.

If the updated parameters to be obtained at point b) are not available, generate your own random values for those parameters.

d) **0.5p** What is the loss function for the testing data point: $x^{(t)} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$, $y^{(t)} = 1$ for the initial model and for the model after one training epoch. Compare the two values. Is the training process convergent from the testing data point of view?