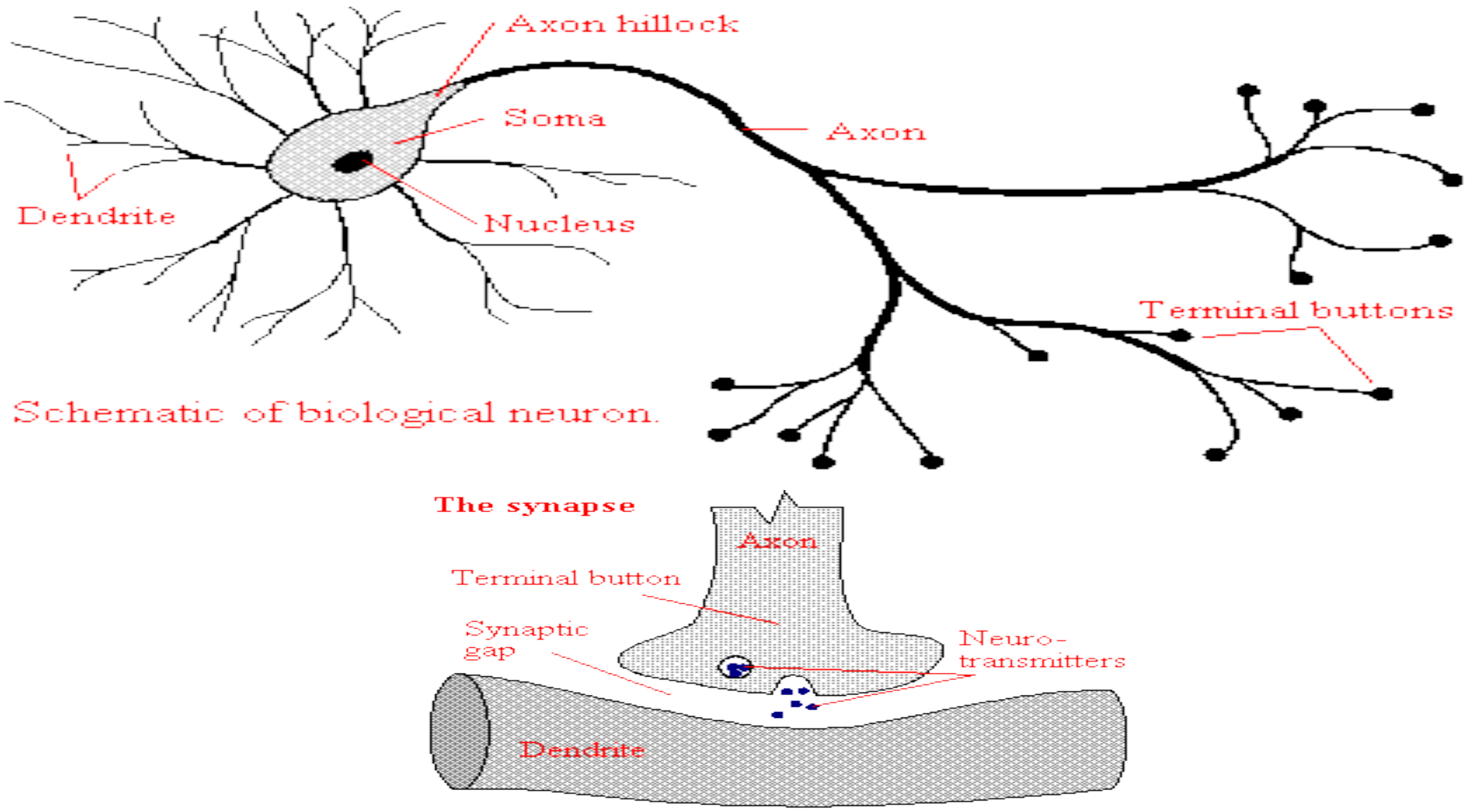
Processamento e Análise de Imagens

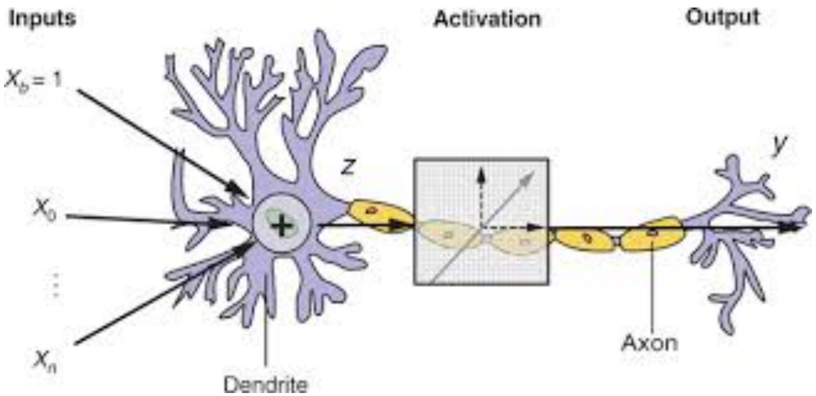
The Perceptron

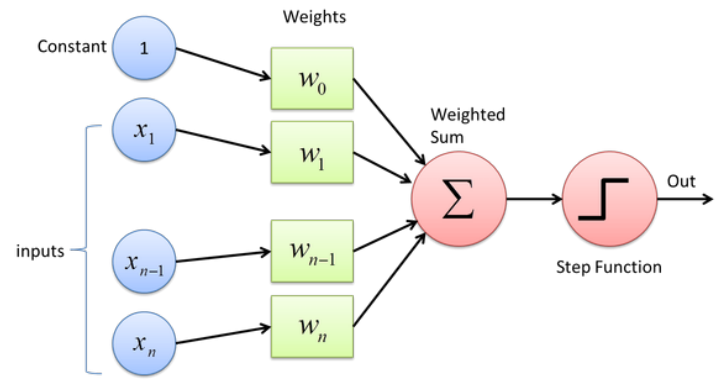
Prof. Alexei Machado
PUC Minas

The Neuron


Axon hillock
Soma
Axon
Dendrite
Nucleus
Terminal buttons
Schematic of biological neuron.
The synapse
Axon
Terminal button
Synaptic gap
Neuro-transmitters
Dendrite

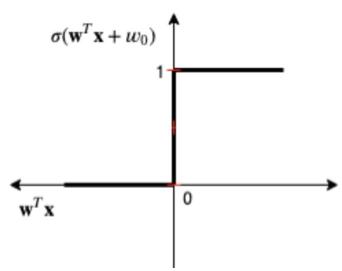
The Perceptron (Rosenblatt 1958)





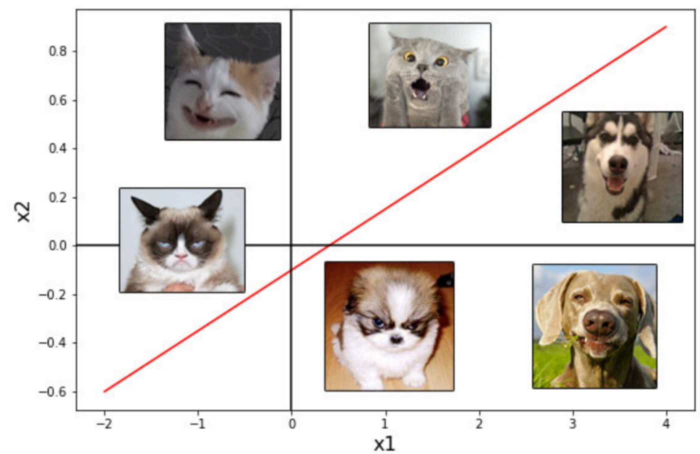
$$z = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = b + \mathrm{w}^T \mathrm{x}$$

$$a = \hat{y} = \sigma(z)$$



The Perceptron (Rosenblatt 1958)

- The perceptron is a binary classifier: y=1 if z>0 otherwise y=0
- The perceptron defines a linear decision function



- Example with 2 variables: the OR operator
- b= -1 (bias)

$$z = -1 + 2x_1 + 3x_2$$

The Perceptron (Rosenblatt 1958)

- Solutions also can be found for the AND and NOT operator
- What about XOR?

The Perceptron (Rosenblatt 1958)

- How do we find a feasible solution?

1. Initialize W
2. Repeat until W is stable (convergence)

2.1. For each sample (x,*y*) in the dataset

2.1.1 Compute $\hat{y} = \sigma(b + w^T x)$

2.1.2. If $\hat{y} \neq y$ then adjust each weight w_i so that $\hat{y}$ *gets closer to y*

- I.e. We backpropagate the output error in order to get a better estimate of the weights
- If the classes are linearly separable, the algorithm will converge, otherwise it will not!

The Perceptron (Rosenblatt 1958)

Remarks:

- A Sum of Squared error fuction can be used to evaluate convergence (Be careful to overfitting!)
- If the classes are linearly separable, the algorithm will converge, otherwise it will not! (Duda, Hart e Stork)
- The algorithm finds ANY solution that makes it converge. The SVM is an evolution of the perceptron that finds a decision function with maximum separability (Krauth e Mezard, 1987)
- If some input variable is useless its weight should have a small magnitude
- If we increase the number of input variables, linear separabiity may be achieved (not always, of course!)

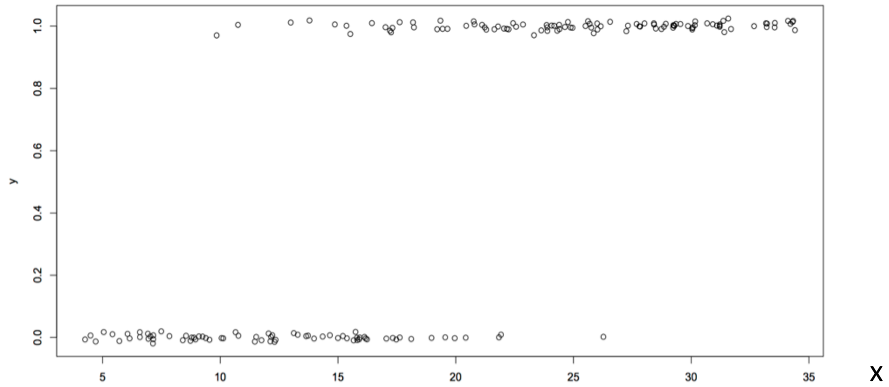
The Perceptron (Rosenblatt 1958)

Remarks:

- In 1969 Minsky and Pappert showed the weakness of the Perceptron to solve the XOR problem
- The research on Connectionism slowed down
- The perceptron model then receives 2 modifications: a different activation function and additional layers.

Logistic Regression

- In many cases, the behavior of a variable x does not change drastically from one class to the other:

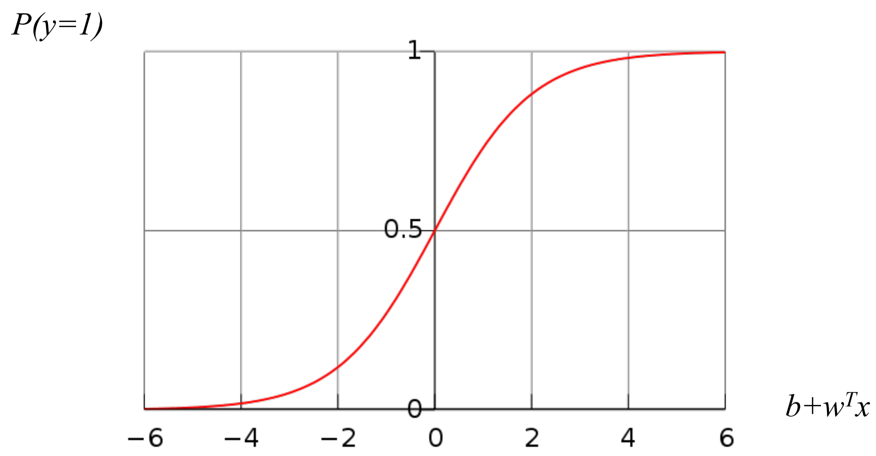


- Therefore we want the output of the classifier to give the probability of the class instead of being a 0/1
- The probability however is not properly defined by a linear function!

Logistic Regression

- We model the log of the odds ratio as a linear function, from which we get the sigmoid ativation function:

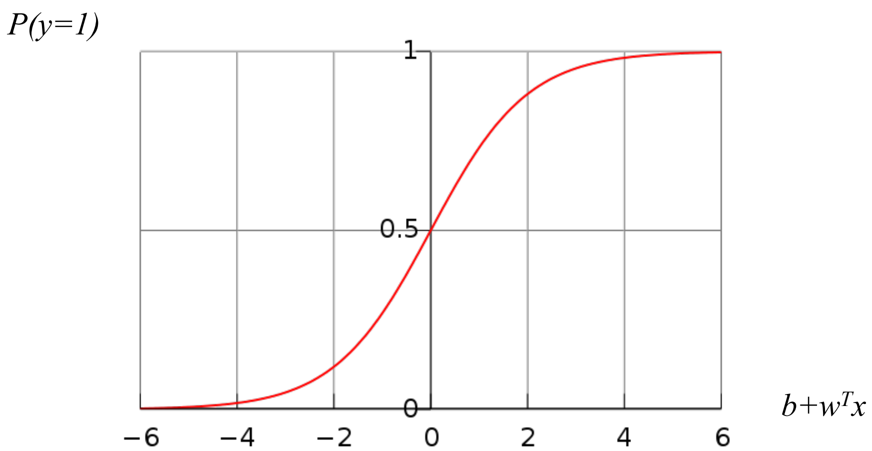
$$\ln \frac{P(y=1)}{1-P(y=1)} = b + \mathrm{w}^T \mathrm{x}$$



Logistic Regression

- We model the log of the odds ratio as a linear function, from which we get the sigmoid ativation function:

$$P(y=1) = \frac{1}{1+e^{-(b+\mathrm{w}^T\mathrm{x})}}$$



Logistic Regression

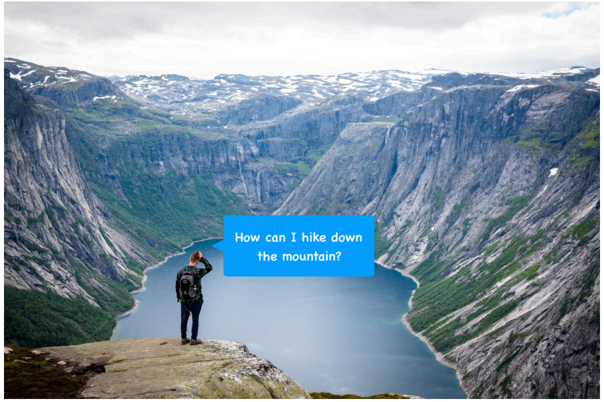
- How to adapt the perceptron's learning algorithm to the new activation function?
  - We need to properly update the weights
  - We need to compute a loss function and know when to stop iterating
- For the logistic activation function, the SSE loss function does not work well
- Since P(y=1|x)=y^ and P(y=0|x)=1-y^ ,we minimize the *cross-entropy loss function:*

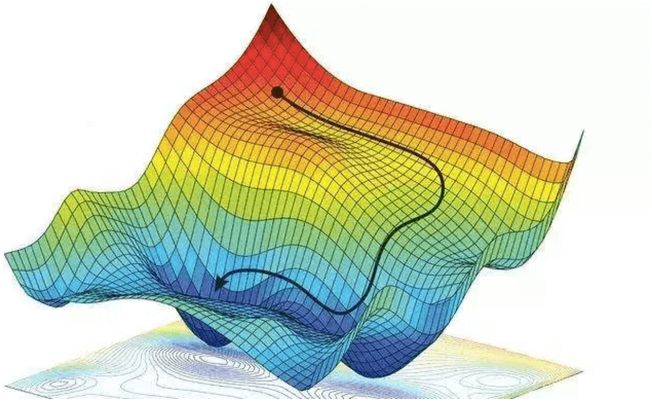
$$L(y,\hat{y}) = -\ln P(y\,|\,\mathrm{x}) = -\ln(\hat{y}^{y}(1-\hat{y})^{1-y}) = -(y\ln\hat{y} + (1-y)\ln(1-\hat{y}))$$

$$J(\mathrm{w},b) = \tfrac{1}{m}\sum_{i=1}^{m} L(y^{(i)},\hat{y}^{(i)})$$

Gradient Descent

- A man is lost in the mountains and is trying to get down to the road, which is in the lowest pat of the region.
- There is heavy fog such that visibility is extremely low, therefore, the path down the mountain is not visible.
- So the best he can do is to take the path that takes him to the lowest possible place from his current position.
- If we think that the geometry of the region gives altitude as a function of position, the gradient of it will point to the (locally) best path!





Gradient Descent

- In our case, the function that tells us how far we are from the best solution is the Loss function!
- But what is the gradient of the Loss function?

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

$$\boxed{\partial L(a,y) / \partial a = -\frac{y}{a} + \frac{1-y}{1-a}}$$

$$\boxed{\partial L(a,y) / \partial z = a - y}$$

$$\boxed{\partial L(a,y) / \partial w_i = x_i \partial L(a,y) / \partial z}$$

$$\boxed{\partial L(a,y) / \partial b = \partial L(a,y) / \partial z}$$

Gradient Descent

1. Initialize parameters / Define hyperparameters
2. Loop for num_iterations:
    a. Forward propagation
    b. Compute cost function
    c. Backward propagation
    d. Update parameters (using parameters, and grads from backprop)
4. Use trained parameters to predict labels

Gradient Descent

$J = 0,\ dw_1 = 0,\ dw_2 = 0,\ db = 0$

for i = 1 to m:

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$

$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$

$$db \mathrel{+}= dz^{(i)}$$

$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m$

$db = db/m$

Complexity?

References and acknowledgements

Some of these slides were inspired or adapted from courses and presentations given by Andrew Ng, Camila Laranjeira, Fei-Fei Li, Flávio Figueiredo, Hugo Oliveira, Jefersson dos Santos, Justin Johnson, Keiller Nogueira, Pedro Olmo, Renato Assunção, Serena Yeung.

Reference courses include *Machine Learning* and *Deep Learning* CS230 and CS231 from Stanford University, *Deep Learning* and *Hands-on Deep Learning* from UFMG, *Deep Learning* CS498 from Un. Of Illinois.