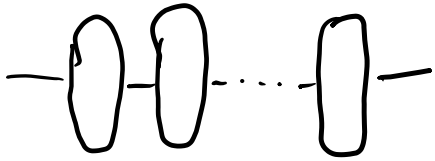
Insper

Machine Learning

Aula 20 – Introdução à redes neurais (III)

2021 – Engenharia
Fábio Ayres <fabioja@insper.edu.br>

- modelo: 
  e tem parametros
- dados: X_{train}, y_{train}
- função de perda: $L(\text{dados}, \text{parâmetros}) = L(\theta)$
  (dados: são fixos; parâmetros: queremos otimizar)
- problema de otimização: $\arg\min_{\theta} L(\theta)$
- algoritmo de otimização:
  - Gradient descent: $\theta^{(i+1)} = \theta^{(i)} - \eta \cdot \nabla_{\theta} L(\theta)$
    - simples: só depende de calcular $\nabla_{\theta} L(\theta)$
      ↳ auto diff.

- 2 APIs: Sequential e Functional

+ simples

+ flexível

- Formato dos dados: tensores → "matrizes" n-dimensionais

rank (posto)

  - número: tensor de rank zero → x: não tem índice
  - vetor: tensor de rank 1 → x[i]
  - matriz: tensor de rank 2 → x[i, j]
  - tensor de rank k:

$$x[i_1, i_2, \ldots, i_k]$$

- Dimensões do tensor tem significado especial:

X.shape => (100, 28, 28)

100: número de amostras num batch

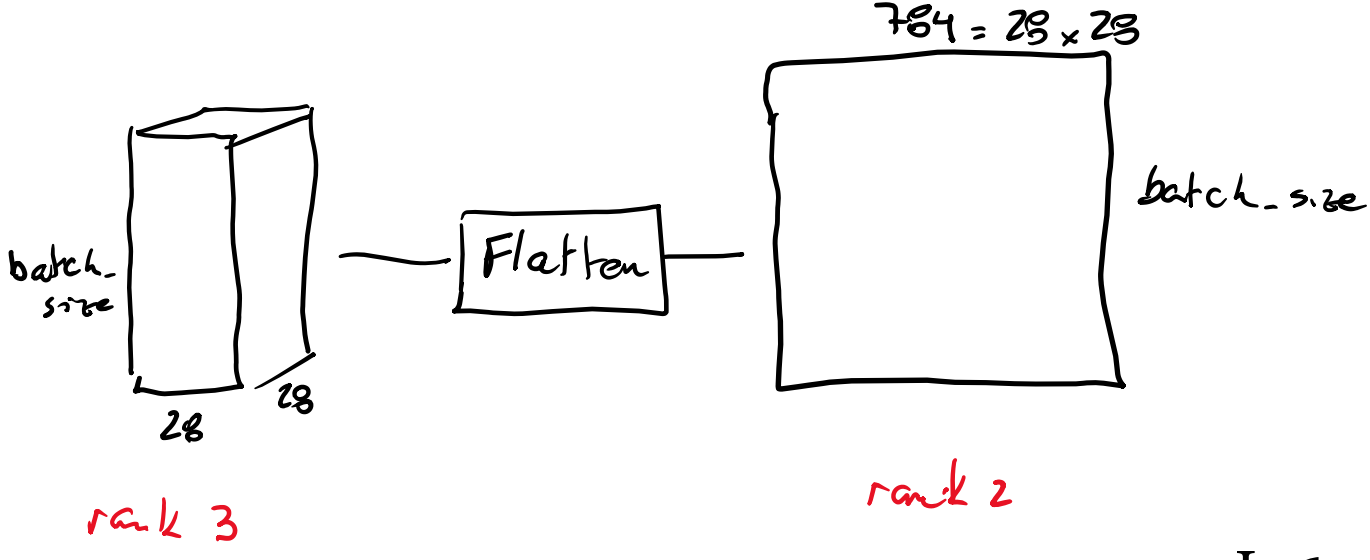
28, 28: depende da camada

Exemplos:

- Dense: requer (batch_size, num_features) — rank 2
- Conv2D: requer (batch_size, linhas, colunas, canais)
- LSTM: (batch_size, tempo, features)
- TimeDistributed(model): (batch_size, tempo, ---)

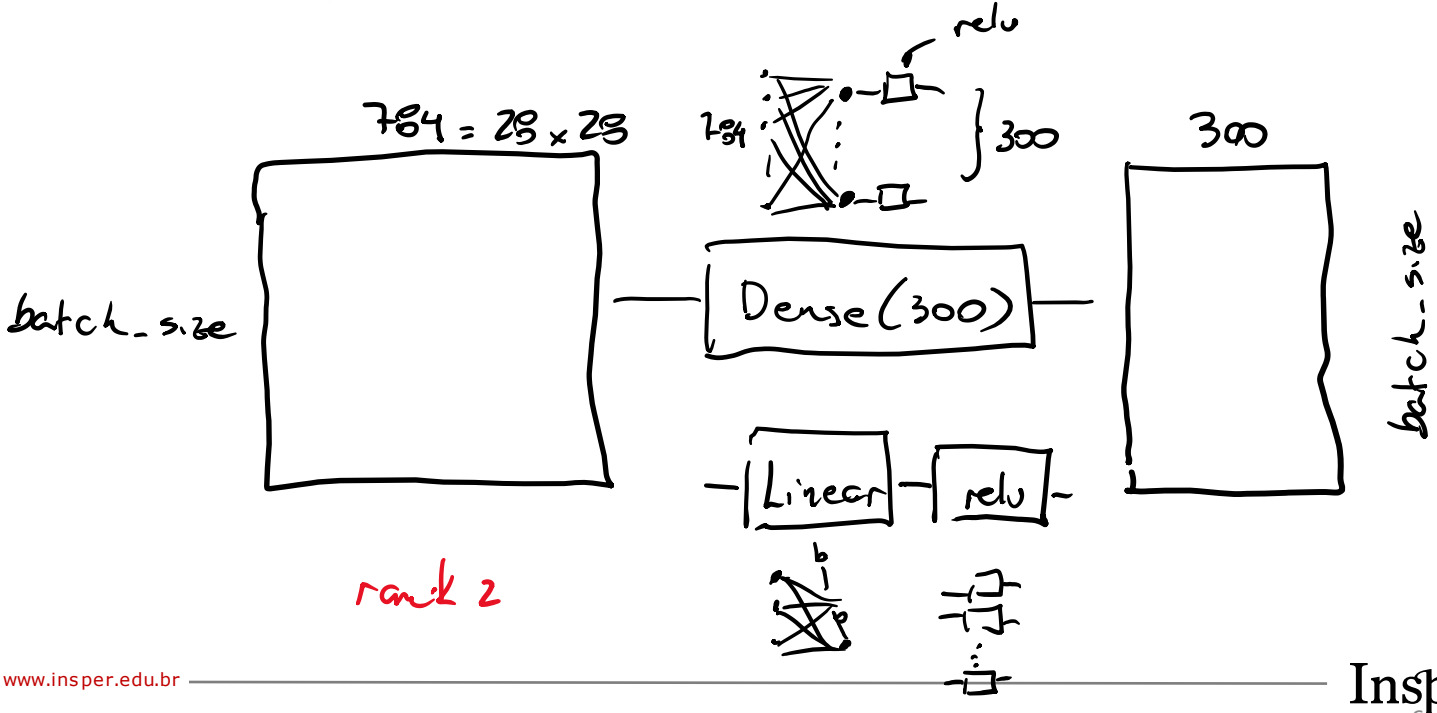
Flatten:

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax"),
])
```



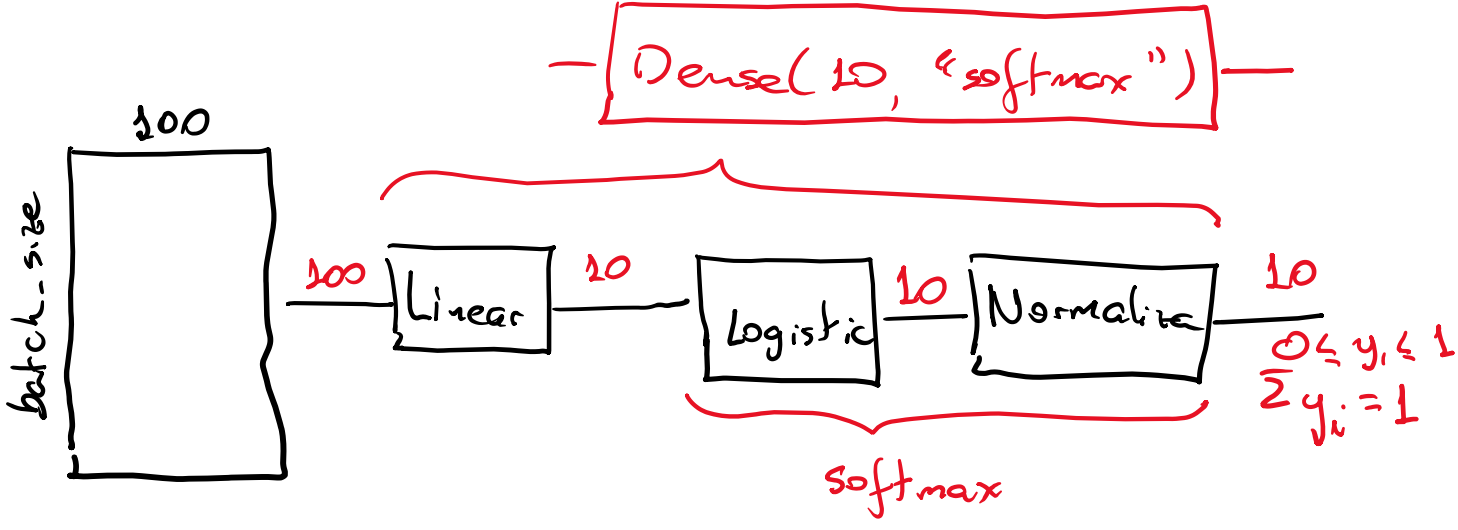
Dense:

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax"),
])
```

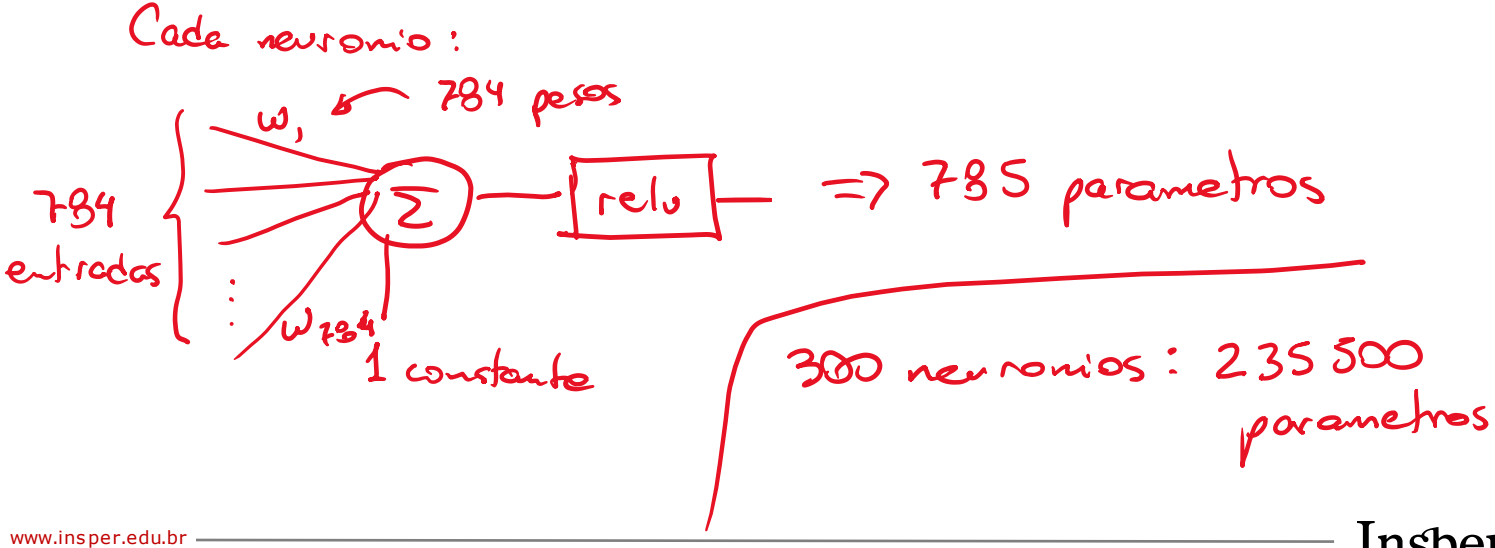


Dense:

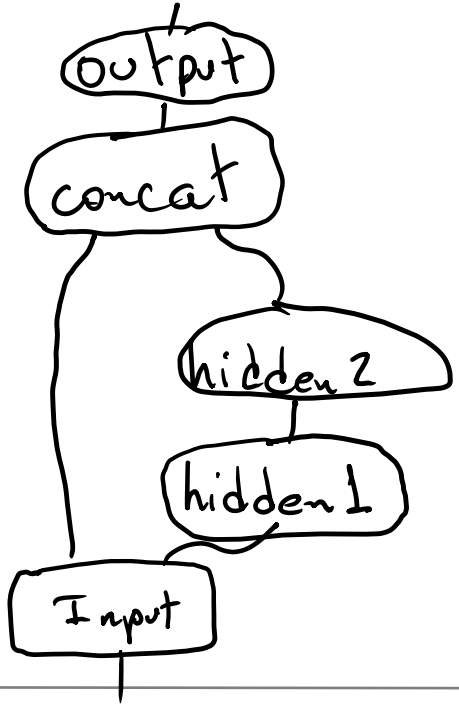
```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax"),
])
```



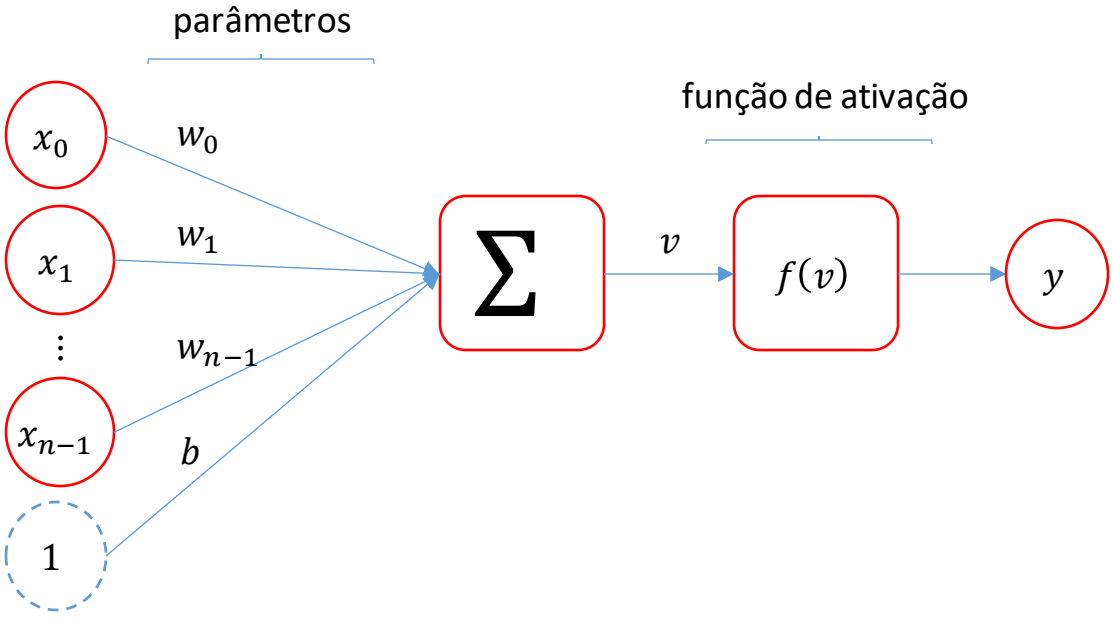
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 300)               235500

 dense_1 (Dense)             (None, 100)               30100

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```



```
input_ = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input_)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input_], outputs=[output])
```



Neurônio artificial



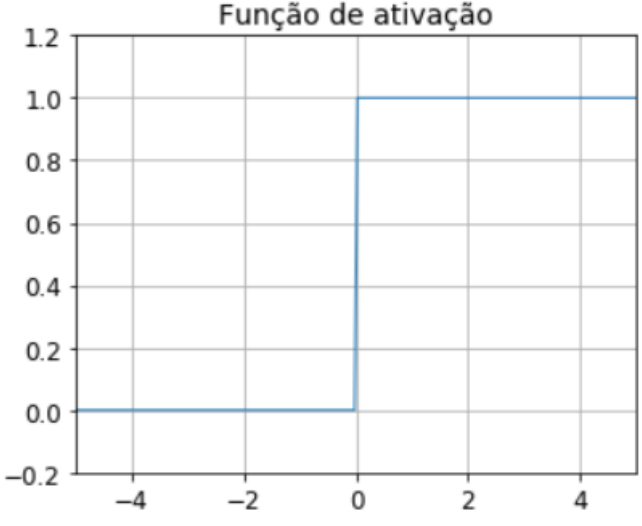
Funções de ativação

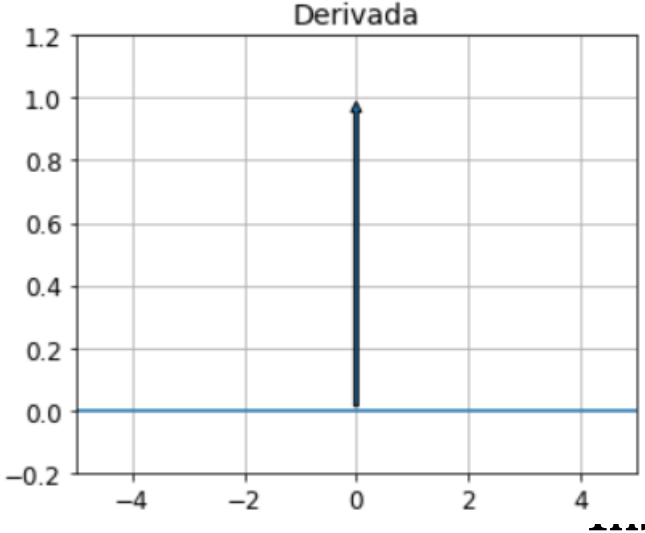
- Conferem o carater não-linear da rede neural
  - Senão seria apenas uma transformação afim (linear + constante)

- Algumas funções comuns:
  - Função identidade: para quando não queremos a não-linearidade mesmo
  - Anos 60: Degrau e sinal
  - Anos 80: Sigmoides – logistica e tangente hiperbólica
  - Anos 2000: Rectified linear units – relu, elu, selu, gelu, etc...

Função degrau (Heaviside)

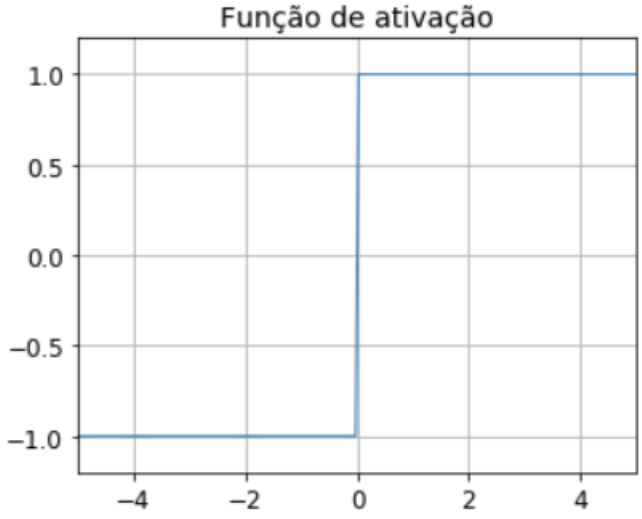
- Trivial de calcular
- Problema: a derivada é inútil (este problema aparece muito nas redes neurais modernas!)

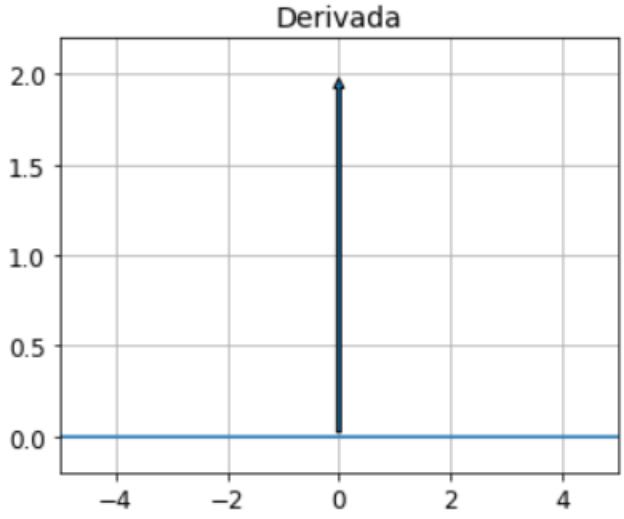




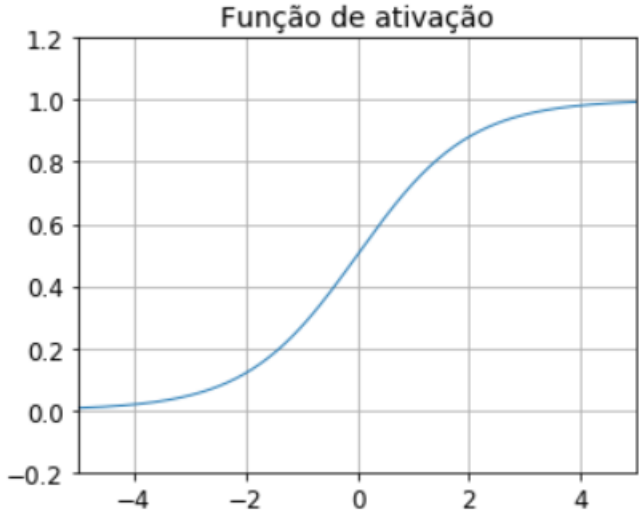
Função sinal

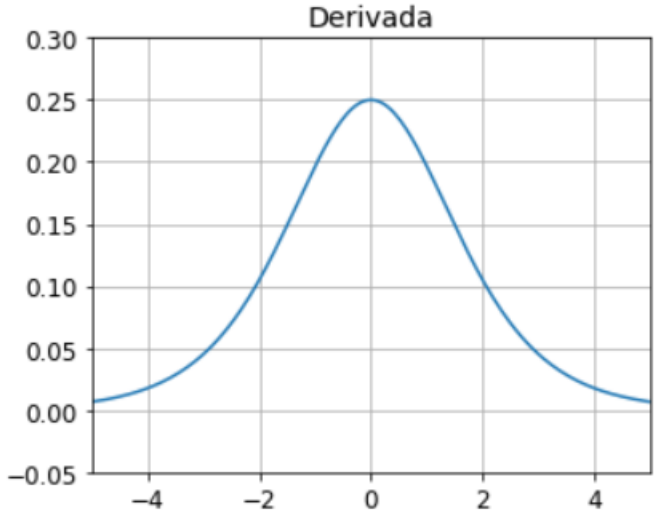
- Trivial de calcular
- Mesmo problema da derivada nula





Função logística



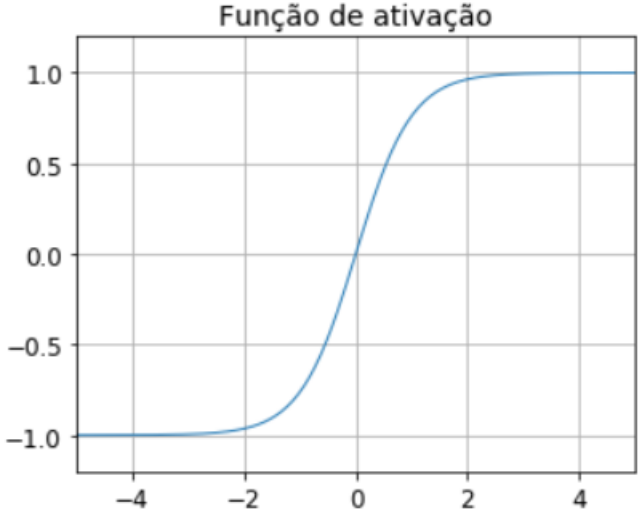


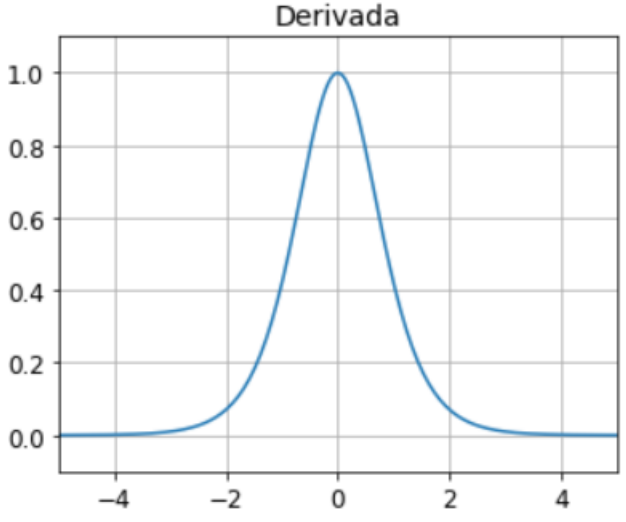
Função logística

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Contínua e derivável
- Pesada para calcular
- Não sofre de descontinuidade na derivada
- Sofre do problema das derivadas quase-nulas longe da origem

Função tangente hiperbólica



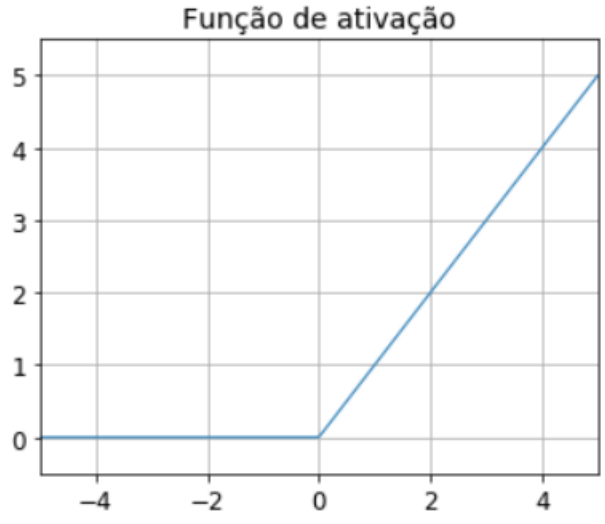


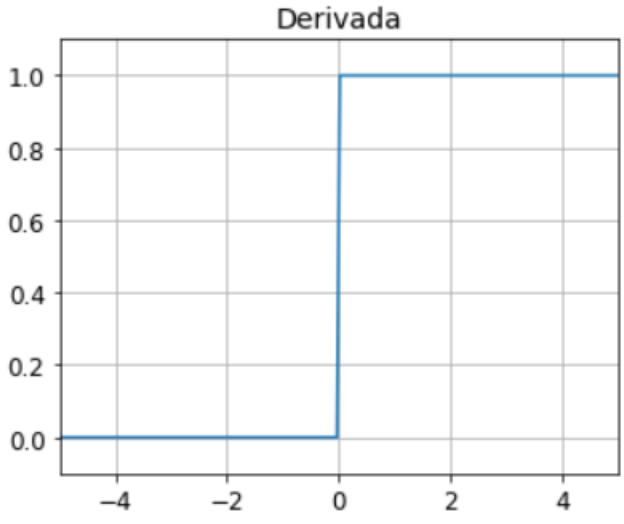
Função tangente hiperbólica

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Contínua e derivável
- Pesada para calcular
- Não sofre de descontinuidade na derivada
- Também sofre do problema das derivadas quase-nulas longe da origem

Rectified linear unit



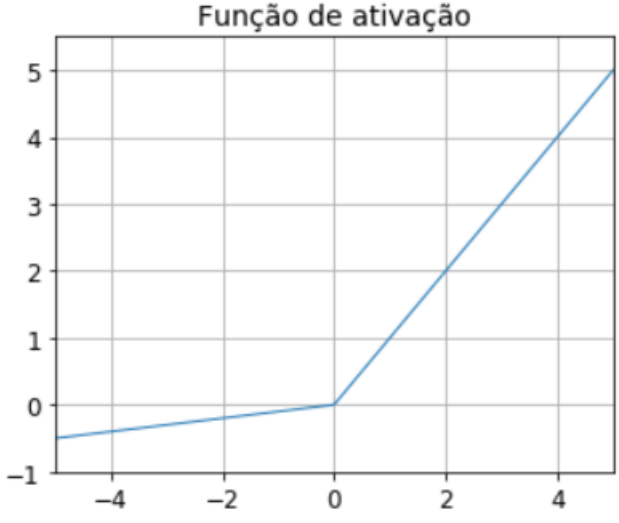


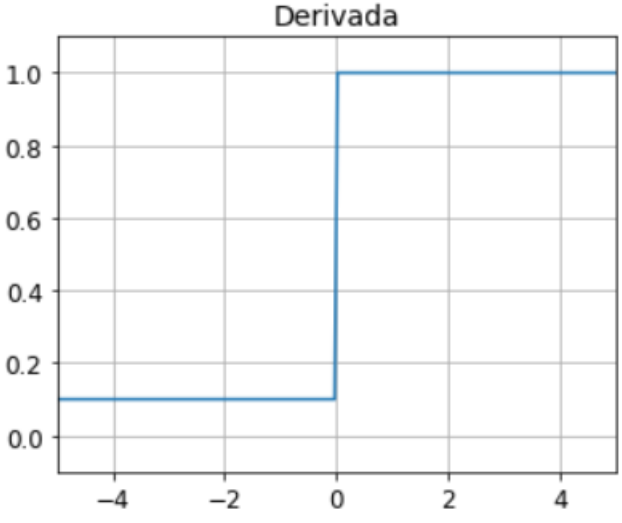
Rectified linear unit (relu)

$$f(x) = f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- Trivial de calcular
- Sofre de derivadas nulas apenas de um lado (isso já ajuda)
- Sofre de descontinuidade na derivada
  - Isso pode fazer o *gradient descent* ficar "oscilando" perto da origem!

Leaky relu


Função de ativação
5
4
3
2
1
0
−1
−4
−2
0
2
4

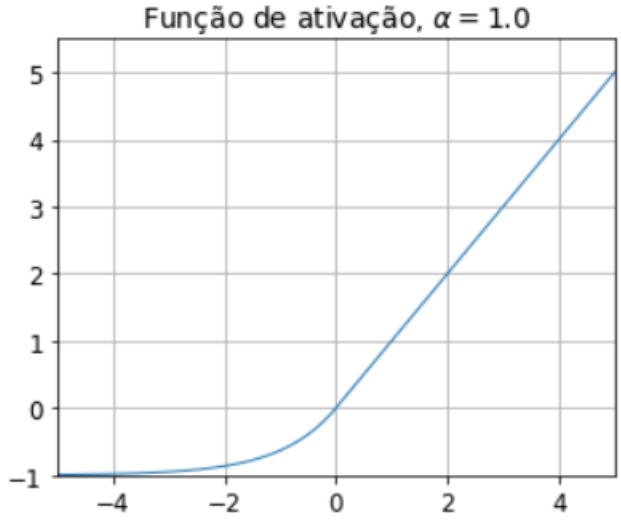

Derivada
1.0
0.8
0.6
0.4
0.2
0.0
−4
−2
0
2
4

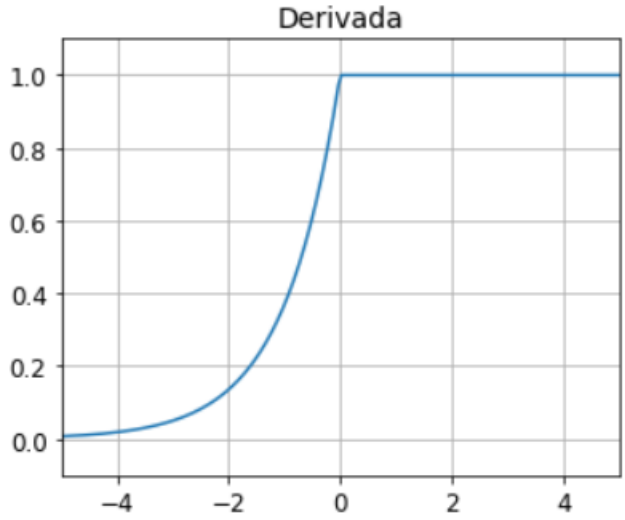
Leaky relu

$$f(x) = f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

- Trivial de calcular
- Não tem derivadas nulas
- Sofre de descontinuidade na derivada

Exponential linear unit (elu)



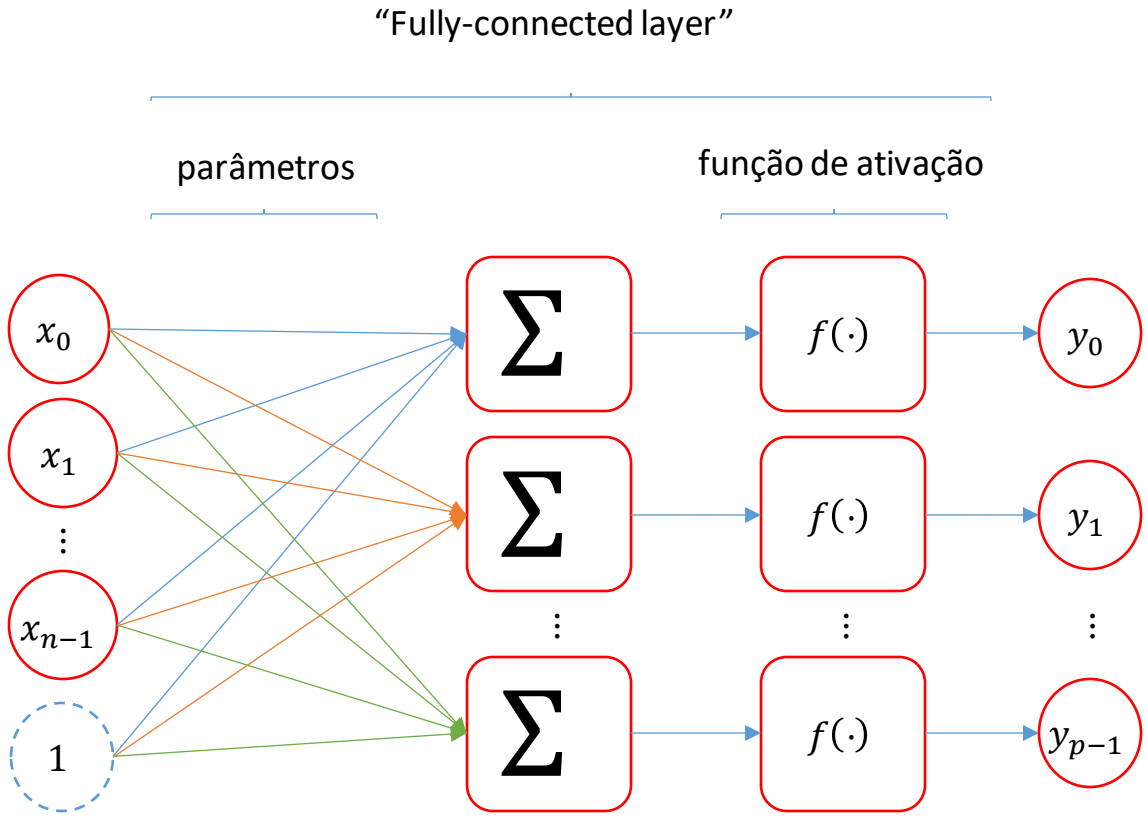


Exponential linear unit (elu)

$$f(x) = f(x) = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$$

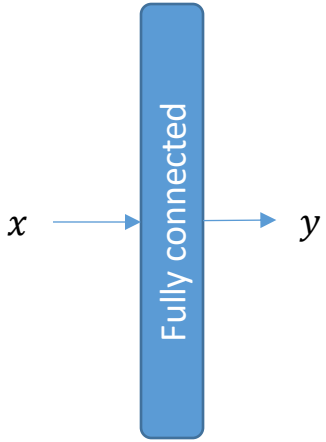
- Pesado de calcular
- Não tem derivadas nulas
  - Mas longe da origem a derivada é quase-nula
- Se $\alpha = 1$ não sofre de descontinuidade na derivada

Camada (layer) de neurônios

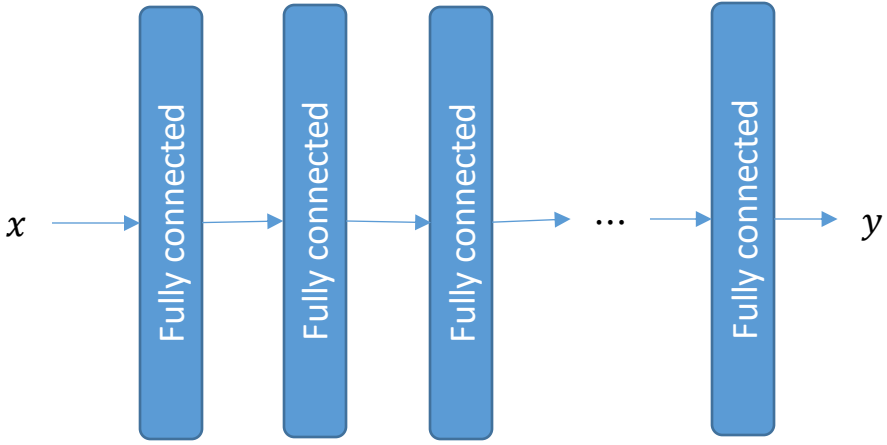


Camada de neuronios

- Geralmente representada de modo simples:



Rede neural multicamadas



Softmax

Insper