AC – Aprendizagem Computacional / Machine Learning

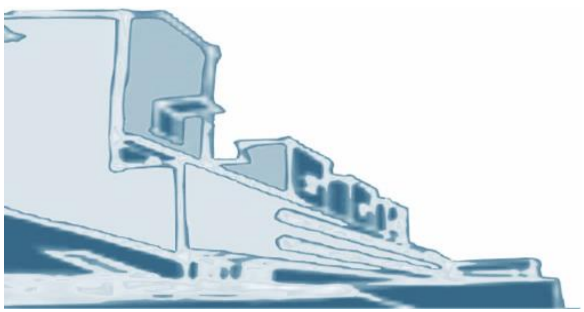
P4b – Multilayer neural networks

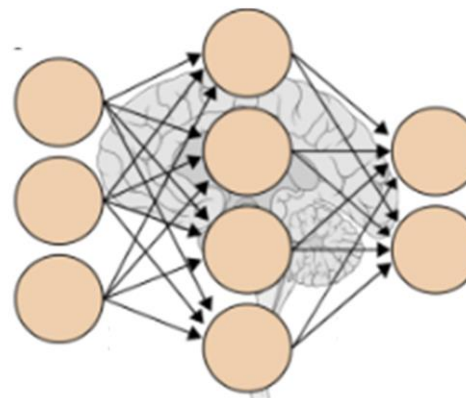
Jorge Henriques

jh@dei.uc.pt

Departamento de Engenharia Informática

Faculdade de Ciências e Tecnologia





1 2 9 0

UNIVERSIDADE D COIMBRA

dei engenharia informática

Contents

- MLNN - Multi Layer neural networks

  - Concepts

  - Implement **BackPropagation** from scratch
    - Regression - Approximate a non-linear function

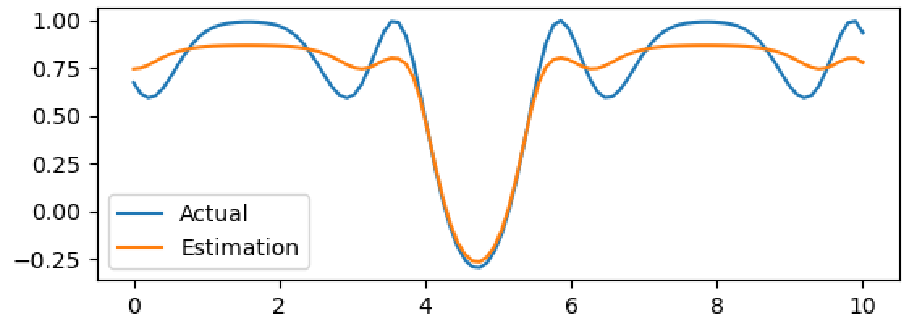
  - Use **python/scikit learning** functionalities
    - Classification
    - Regression

  - Evaluation the performance of the MLNN classifier/regressor
    - SE, SP, F1score (*classification*)
    - MSE, R2 (*regression*)

Contents

Datasets

- 1| Function approximation
  - Simple problem – regression
  - P4_function.csv



- 2| Cardiac Risk
  - Classification
  - P4_cardiacRiskClassification.csv
  - Output = {0,1} – **{event, NO event}**

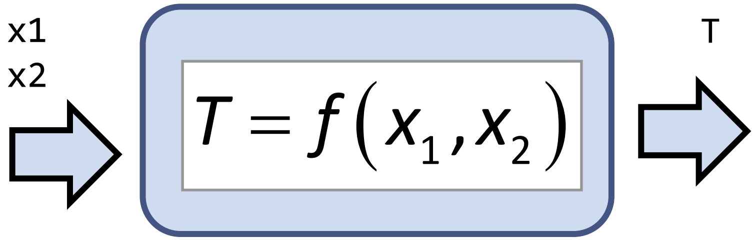
- 3| Cardiac Risk
  - Regression
  - P4_cardiacRiskRegresssion.csv
  - Output = [0..1] – **probability to have an event**

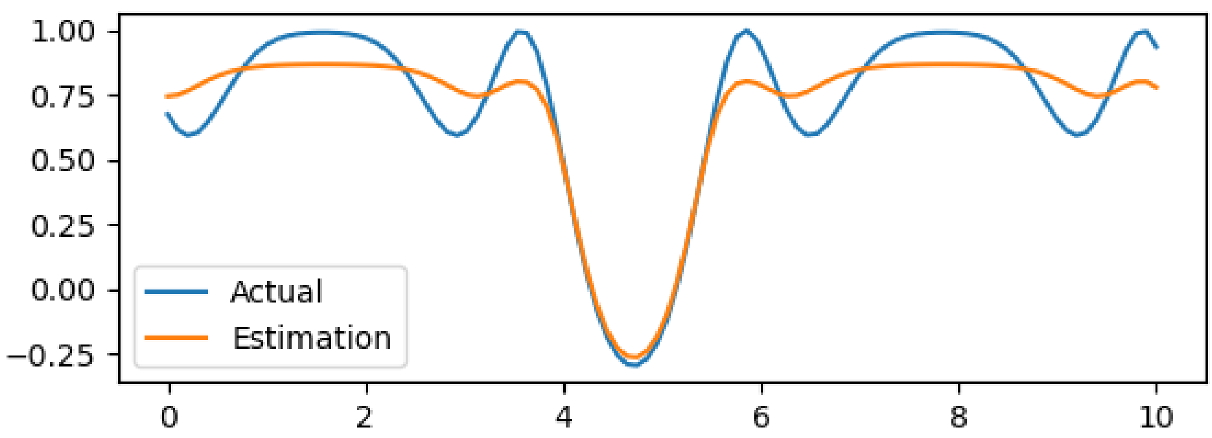
1| Datasets – function approximation

- Non-linear function

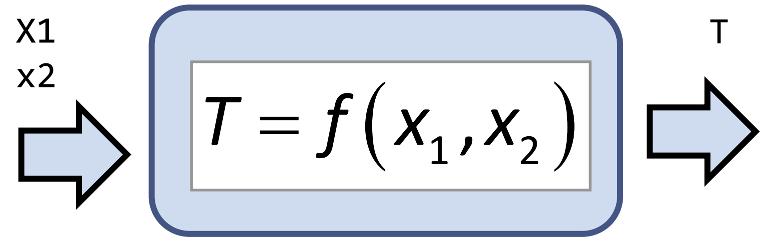
```
t1= 0:0.1:10
x1= sin(t1);
x2= sin(t).cos(t)^22;
T =  x1 + 2.4*x2;
```

P4_function.csv

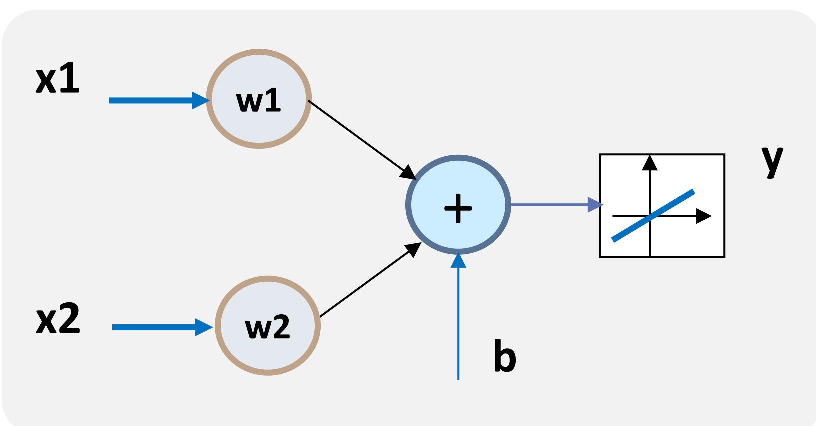


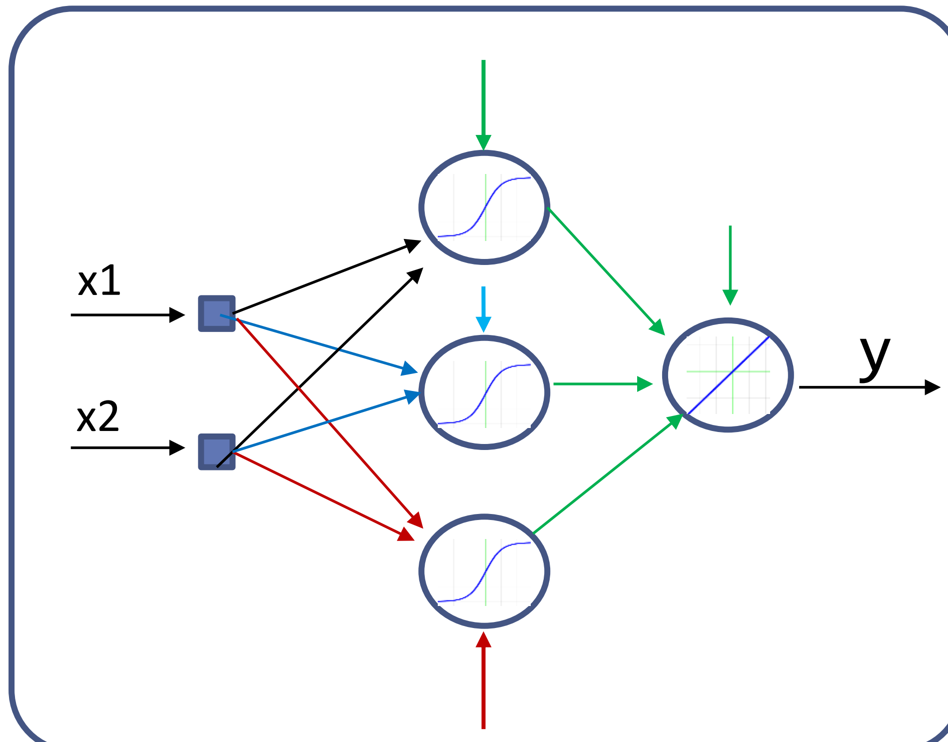


1| Function approximation



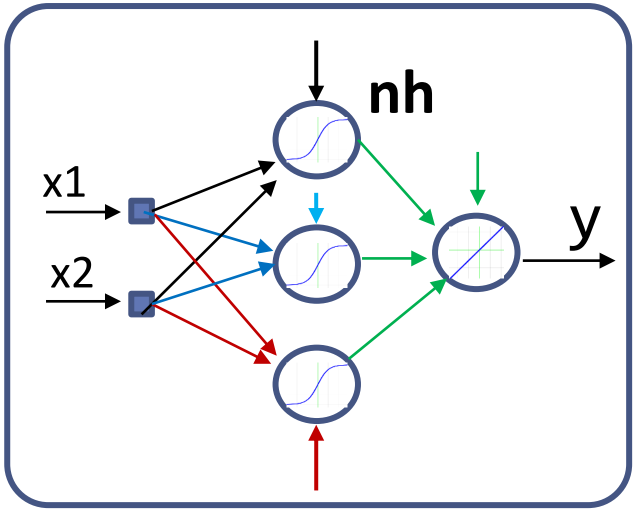
- Two possible structures
  - 1. Adaline : one layer
  - 2. MLNN : hidden layer
- Learning:
  - **1. RMLSE**
  - **2. Backpropagation**





- **2| Datasets - Cardiac Risk**
  - Use of scikit learning functionalities

- Structure: MLNN – one hidden layer
- Activation functions
  - Hidden layer – sigmoidal (nh neurons)
  - Output layer – linear (one neuron)

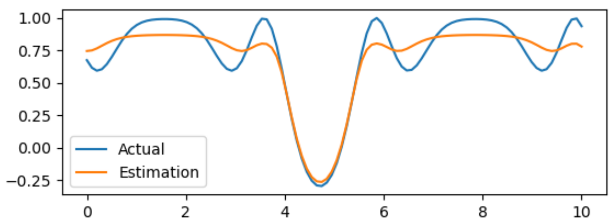


- Classification T={0,1}
- Regression T=[0..1]

Contents

1| Function approximation

P4_function.csv



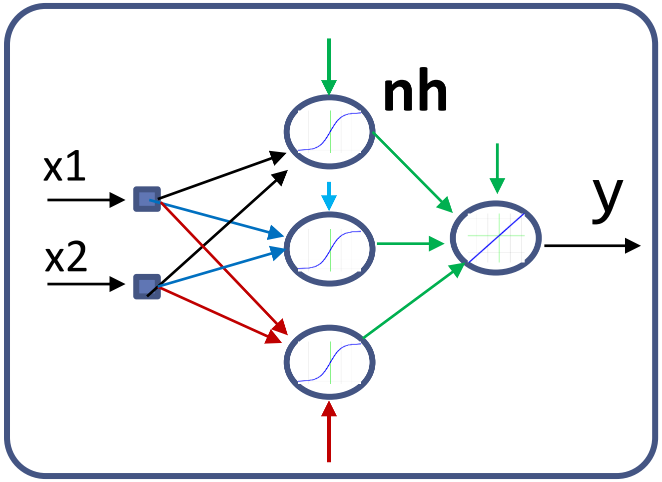
1.1| Structure

Activation function

- Sigmoidal / Linear

Number of hidden layers

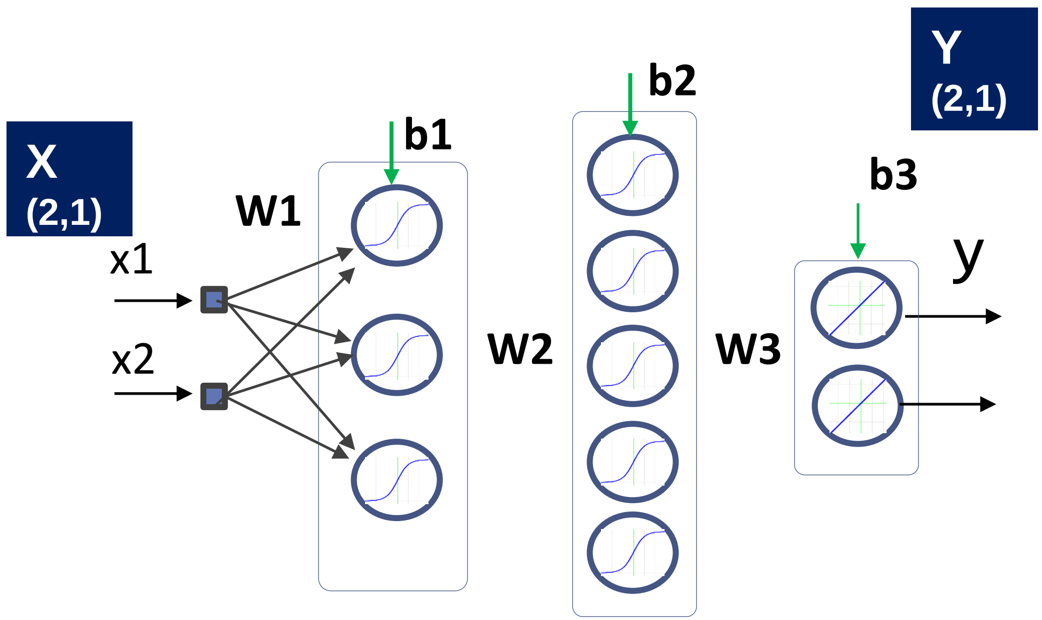
- nh = ?



1.2| Learning: Back Propagation

- Perform the computations by hand !

- **1| BP: example**
  - FORWARD



W1=(3,2)
b1=(3,1)

W2=(5,3)
b2=(5,1)

W3=(2,5)
b3=(2,1)

$$a^0 = x$$

$$n^1 = W^1 a^0 + b^1$$ (3,2)(2,1)+(3,1)

$$a^1 = f^1(n^1)$$ (3,1)

$$n^2 = W^2 a^1 + b^2$$ (5,3)(3,1)+(5,1)

$$a^2 = f^2(n^2)$$ (5,1)

$$n^3 = W^3 a^2 + b^3$$ (2,5)(5,1)+(2,1)

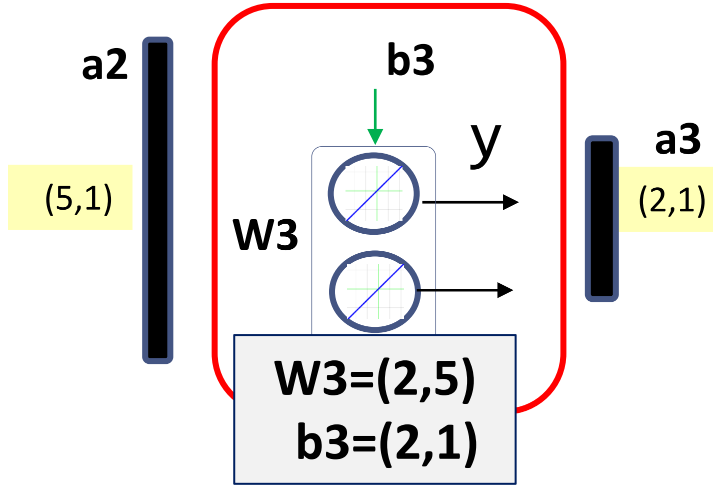
$$a^3 = f^3(n^3)$$ (2,1)

$$y = a^3$$

- **Last layer** (m=3)

$$n^3 = W^3 a^2 + b^3$$
$$a^3 = f^3(n^3)$$

$$e = e^3 = t - a^3 = t - y$$



$$\dot{f}(n^3) = df^3(\cdot)$$

$$s^3 = e^3 \otimes df^3(\cdot)$$

(2,1)⊗ (2,1)

$$\frac{dE}{dw^3} = \frac{d\left(e^3\right)^2}{dw^3} = -2\, e^3\, \dot{f}^3(\cdot)\left(a^2\right)^T$$

$$dW^3 = s^3\left(a^2\right)^T$$

(2,1) (1,5)

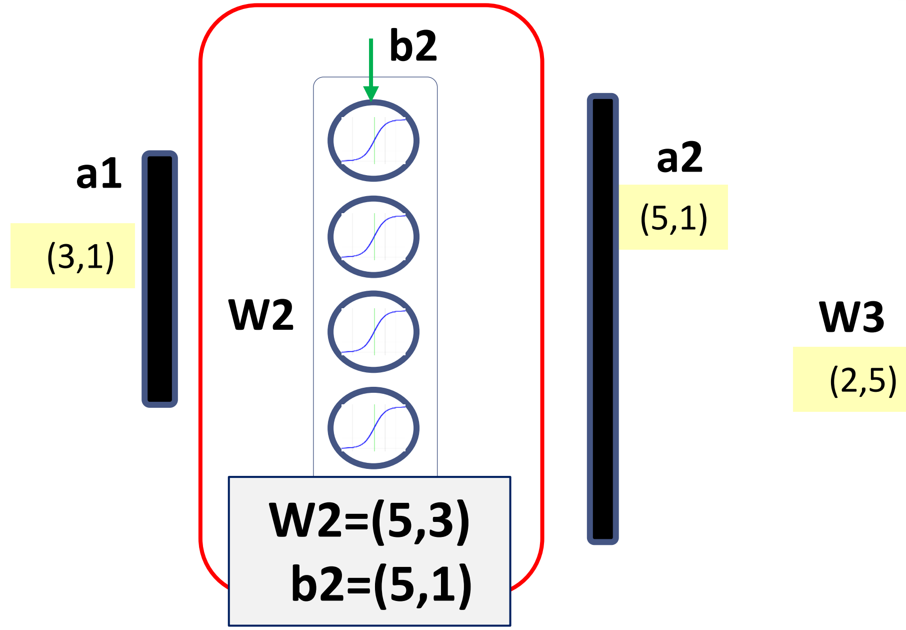
$$\frac{dE}{db^3} = \frac{d\left(e^3\right)^2}{db^3} = -2\, \dot{f}^3(\cdot) e^3$$

$$db^3 = s^3$$

(2,1)

- **Hidden layer** (m=2)

$$n^2 = W^2 a^1 + b^2$$
$$a^2 = f^2(n^2)$$



$$s^{m-1} \equiv \left(w^m\right)^T s^m \dot{f}(n^{m-1})$$

$$\dot{f}(n^2) = df^2(\cdot)$$

(5,1)

$$dW^2 = s^2 \left(a^1\right)^T$$

(5,1) (1,3) = (5,3)

$$s^2 = \left(w^3\right)^T s^3 \otimes df^2(\cdot)$$

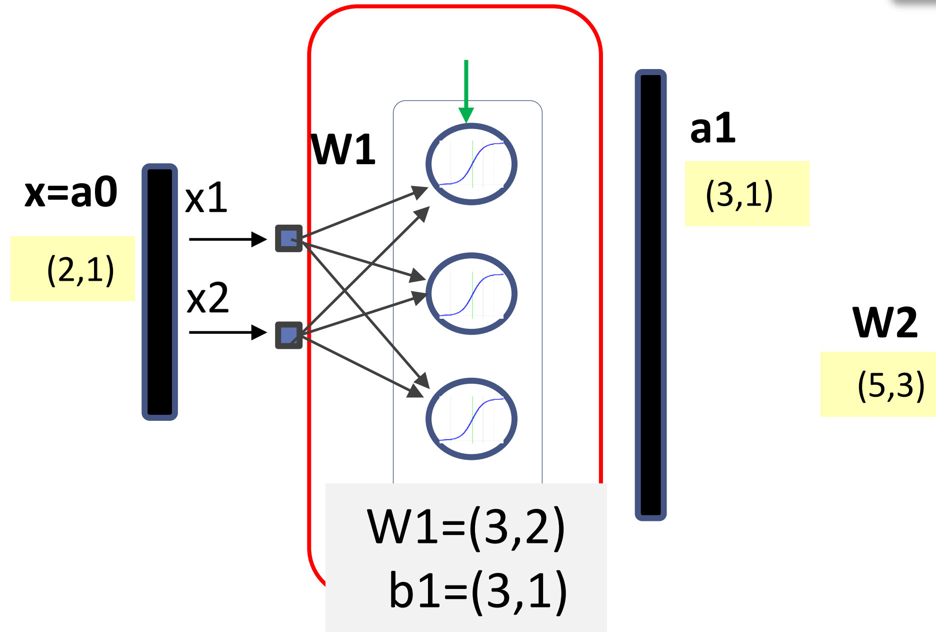
((5,2)(2,1)) $\otimes$ (5,1)

$$db^2 = s^2$$

(5,1)

- **First layer** (*m*=1)

$$n^1 = W^1 a^0 + b^1$$
$$a^1 = f^1(n^1)$$



$$s^{m-1} \equiv \left(w^m\right)^T s^m \dot{f}(n^{m-1})$$

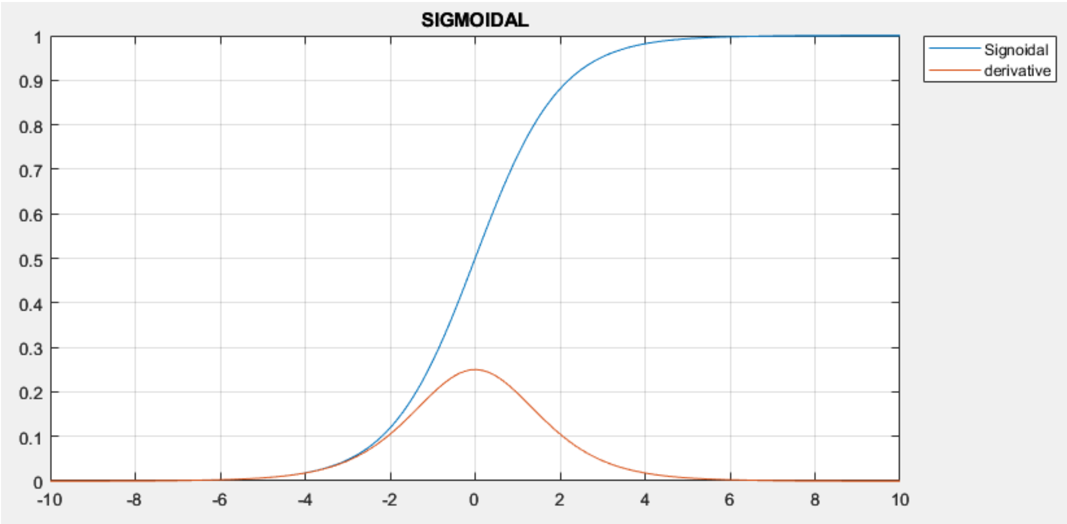
$$\dot{f}(n^1) = df^1(\cdot)$$ (3,1)

$$dW^1 = s^1 \left(a^0\right)^T$$ (3,1) (1,2) =(3,2)

$$s^1 = \left(w^2\right)^T s^2 \otimes df^1(\cdot)$$ ((3,5)(5,1)) ⊗ (3,1)

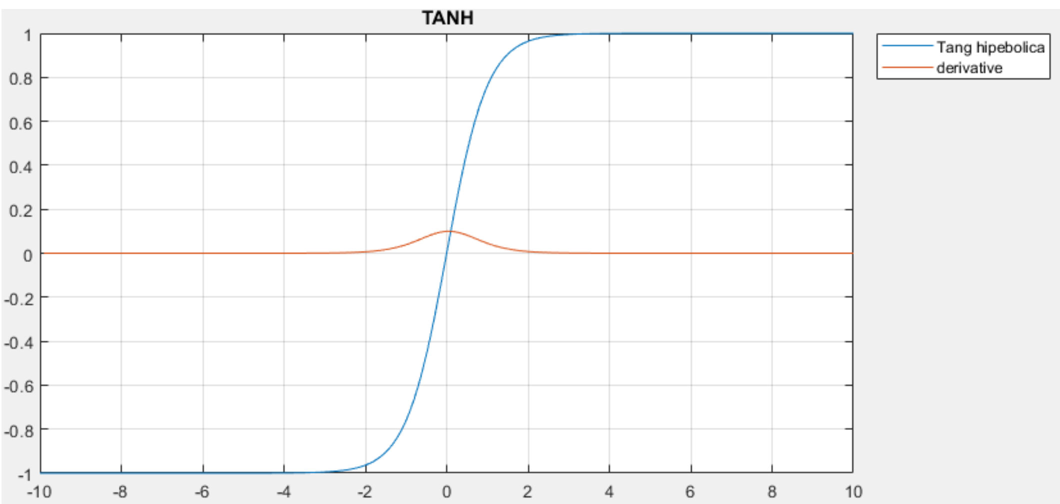
$$db^1 = s^1$$ (3,1)

3| Tasks



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

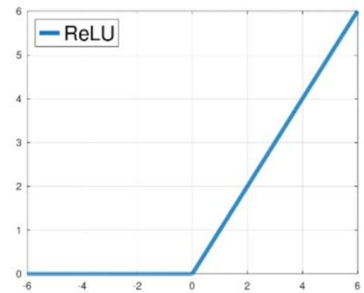
$$df(x)=f(x)\big(1-f(x)\big)$$

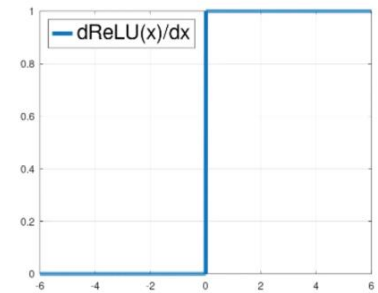


$$f(x)=\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$

RLU

$$f(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \quad f'(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x < 0. \end{cases}$$



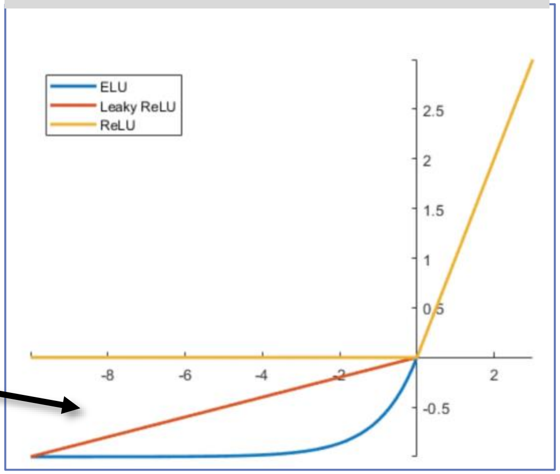


- **Properties**
  - Linear for positive values
  - Non-linear for negative values

- **Advantages**
  - Avoids vanish gradient: unlike sigmodal does not saturate for positive values
  - Simple, fast, efficient

- **Disadvantages**
  - Non-zero centered output
  - Dead neurons: always zero (consistently negative values)

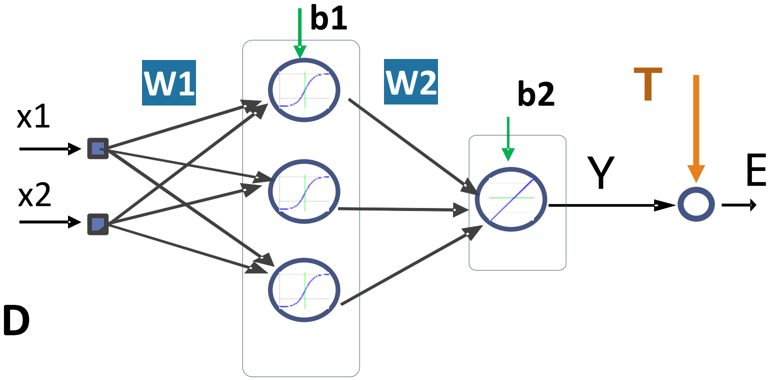
Alternatives



- **1| Function approximation**

  P4_function.csv

```
alfa=0.001
W2=np.random.random((ny,  nh1))
B2=np.random.random((ny,  1))
W1=np.random.random((nh1, nx))
B1=np.random.random((nh1, 1))
MI=np.ones(n,1)
```




```
for numEP in range(EPOCHS):

    #---------------------------- FORWARD
    a0 = X
    a1 = np.dot(W1,a0) + B1
    y1 = sigmoid(a1);            # sigmoidal
    a2 = np.dot(W2,a1) + B2
    y2= a2                       # linear
    #---------------------------- ERROR
    e2 = T - y2
    erro = np.dot( e2, e2.transpose())   # for each EPOCH
    #---------------------------- BACKPROPAGATIN
    dy2 = 1
    S2  = e2*dy2
    dW2 = alfa*np.dot(S2, a1.transpose())
    dB2 = alfa*np.dot(S2, MI)
    W2  = W2+ dW2
    B2  = B2+ dB2
    # . . . .  . . . . .  . . .
   dy1 = y1*(1-y1)
    S1  = np.dot( W2.transpose(), S2)*dy1

    dW1 = alfa* np.dot(S1, a0.transpose())
    dB1 = alfa*np.dot(S1, MI)
    W1  = W1+ dW1
    B1  = B1 + dB1
```

FORWARD

BACKPROPAGATION

$$f_1 = sigmoidal(x)$$

$$\dot{f}_1 = f_1(1 - f_1) = 1 - f_1^2$$

$\otimes$	=	*	element by element
x	= np.dot(.)		multiplication

- **2| Dataset: Cardiac risk /** P4_cardiacRiskClassification.csv

- 2.1| Implement MLNN - classification
  - Structure
    - Number of neurons = ?
    - Activation functions : Hidden layer / Output layer (**sigmoidal/linear**)
  - Training / test data
    - Learning algorithm (**adam**)

```
from   sklearn.neural_network  import MLPClassifier


Xtrain, Xtest, Ttrain, Ttest = train_test_split(X,T,test_size = 0.3,
random_state = 42)


mlp = MLPClassifier(hidden_layer_sizes=(3,2), activation='relu’,
                          solver='adam', max_iter=500)

mlp.fit(Xtrain,Ttrain)
```

- **2| Dataset: Cardiac risk /** P4_cardiacRiskClassification.csv
- 2.2| Evaluation
  - SE, SP, F1score, ?

```
from   sklearn.metrics import confusion_matrix


cm    = confusion_matrix(Ttest, Ytest)
TN, FP, FN, TP = cm.ravel()
SE    = TP/(TP+FN)
SP    = TN/(TN+FP)
F1    = ...
```

- **3| Cardiac risk /** P4_cardiacRiskRegression.csv
- 3.1| Implement MLNN – Regression
  - Structure
    - Number of neurons = ?
    - Activation functions : Hidden layer / Output layer (**sigmoidal/linear**)
  - Training / test data
    - Learning algorithm (**adam**)

```
from   sklearn.neural_network  import MLPRegressor

mlp = MLPRegressor(hidden_layer_sizes=(13,6), activation='relu’,
                    solver='adam', max_iter=500)

mlp.fit(Xtrain,Ttrain)
```

- **3| Dataset: Cardiac risk /** P4_cardiacRiskRegression.csv

- ## 3.2| Evaluation
  - Mean squared Error

```
from   sklearn.metrics  import mean_squared_error

Etrain = mean_squared_error(Ttrain, Ytrain)
Etest  = mean_squared_error(Ttest, Ytest)
```

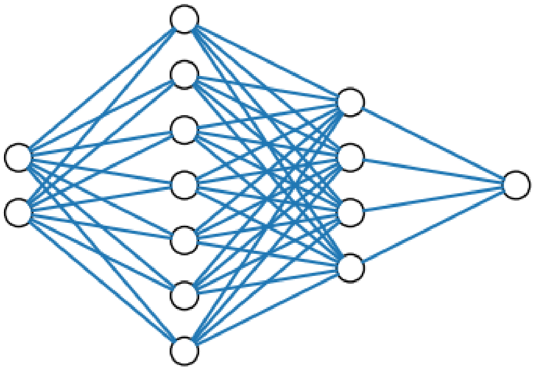
- **3| Cardiac risk /** P4_cardiacRiskRegression.csv
- Output – activation function

```
# Output for regression

#if not is_classifier
        activation_ = 'identity'   ?
        activation_ = 'linear'

# Output for multi class
        activation_ = 'softmax'

# Output for binary class
        activation_ = 'logistic'   ?
        activation_ = 'sigmoid'
```

Contents

- **Multi Layer Neural Networks**
  - Implement by hand the BP algorithm
    - Function approximation
  - Implement a MLNN
    - Classification
    - Regression problem
  - Evaluation
    - SE, SP - classification
    - Error - regression

- Improvements
  - Function approximation
    - 2 hidden layers ?



  - MLNN - Cardiac Risk
    - Activation functions : Hidden layer / Output layer ??
    - Learning algorithm ?
  - Other ideas ?
    - Use of keras / tensorflow (more general tool)

- Scikit / Keras

KERAS=0

```
from   sklearn.neural_network  import MLPClassifier
from   sklearn.neural_network  import MLPRegressor
```

KERAS=1

```
from   keras.models import Sequential
from   keras.layers import Dense
```

```
if KERAS==0:
    if regression:
        model = MLPRegressor( hidden_layer_sizes=(12,8),
                                         activation='relu',
                                      solver='adam', max_iter=3500)
    else:
        model = MLPClassifier(hidden_layer_sizes=(11,4),
                                      activation='tanh',
                                      solver='adam', max_iter=4500)

    model.fit(Xtrain,Ttrain)
```

- Scikit / Keras

```
if KERAS==1:
    model = Sequential()
    if regression:
        model.add(Dense(32,activation='sigmoid'))
        model.add(Dense(18,activation='sigmoid'))
        model.add(Dense( 1,activation='linear’))

        model.compile(   loss='mse', optimizer='adam’,
                         metrics=['mse'])
    else:
        model.add(Dense(33,activation='sigmoid')    )
        model.add(Dense(12,activation='sigmoid')    )
        model.add(Dense( 4,activation='sigmoid')    )
        model.add(Dense( 1,activation='sigmoid’))

        model.compile(   loss='binary_crossentropy’, optimizer='adam’,
                         metrics=['accuracy'])


    model.fit(Xtrain,Ttrain, batch_size=40, epochs=125, verbose=1)
```

- Scikit / Keras

```
history=model.fit(       Xtrain,Ttrain,  validation_split=0.3,
                         batch_size=40, epochs=125,verbose=1)

if KERAS==1:
    #df = pd.DataFrame.from_dict(history.history)

    print(history.history.keys())
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])

    plt.title(' Training error')
    plt.ylabel('Error')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```