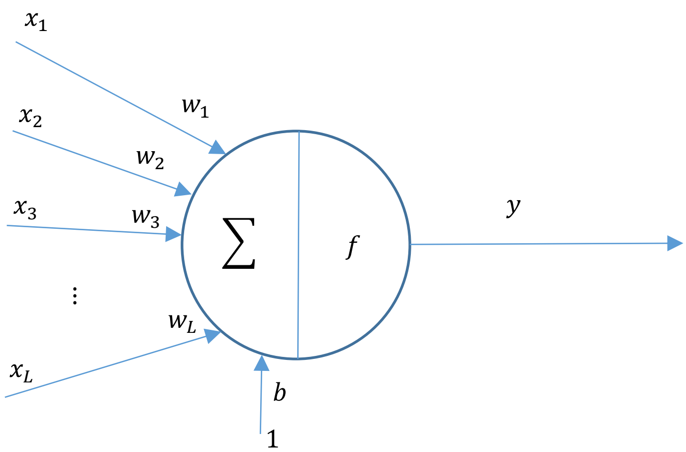
ĆWICZENIE 4

MODEL NEURONU

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z modelem neuronu McCullocha-Pittsa.

Wprowadzenie



Rys. 1. Model neuronu

Sygnał wyjściowy neuronu y określony jest zależnością:

$$y = f\left(\sum_{j=1}^{L} w_j \cdot x_j + b\right), \tag{1}$$

gdzie x_j jest j-tym ($j = 1,2,..L$) sygnałem wejściowym, a w_j- współczynnikiem wagowym (wagą). Ze względu na skrócenie zapisu wygodnie będzie stosować zapis macierzowy do opisu działania neuronu. Niech $\mathbf{x} = \left[x_1, x_2, \ldots, x_L\right]^T$ będzie wektorem sygnałów wejściowych, $\mathbf{w} = \left[w_1, w_2, \ldots, w_L\right]$ - macierzą wierszową wag, a y i b - skalarami. Wówczas

$$y = f(\boldsymbol{w} \cdot \mathbf{x} + b) \tag{2}$$

Ważona suma wejść wraz z przesunięciem często bywa nazywana łącznym pobudzeniem neuronu i w dalszych rozważaniach oznaczana będzie symbolem z

$$z = \sum_{j=1}^{L} w_j \cdot x_j + b. \tag{3}$$

Przebieg laboratorium

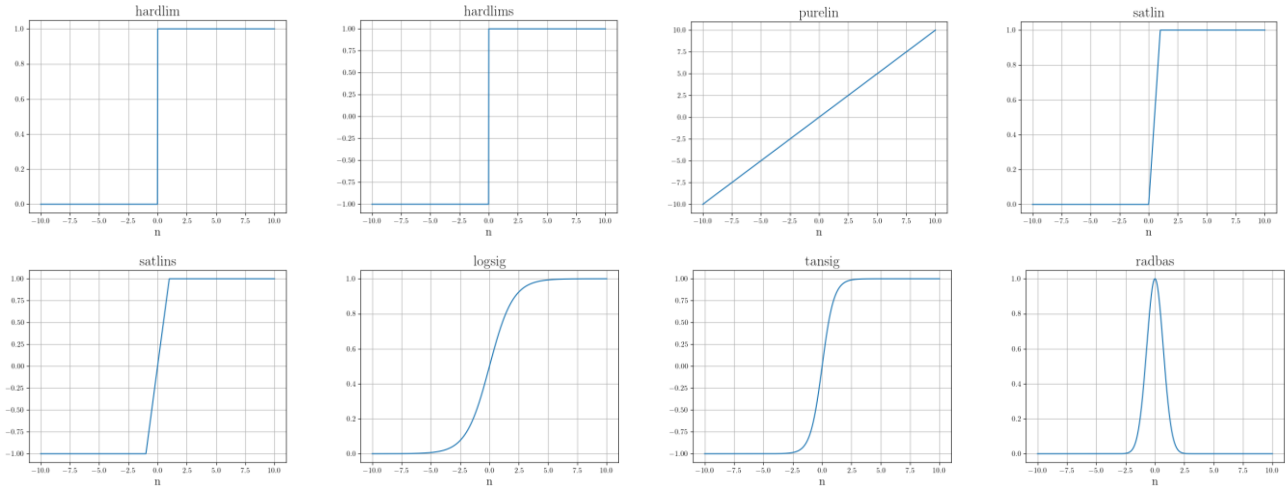
1. Dla pojedynczego neuronu o wagach [.1 .4 -.3 .7], przesunięcia [.5] i wektora sygnałów wejściowych [1 2 3 4] wyznaczyć łączne pobudzenie neuronu.

   Przykładowe rozwiązanie:
   ```
   import numpy as np
   w=np.array([.1, .4, -.3, .7])
   x=np.array([range(1,5)])
   b=.5
   ```

```
z=w.dot(x.T)+b
```

W sprawozdaniu zaproponować trzy inne alternatywne metody wyznaczenia łącznego pobudzenia neuronu.

2. Zapoznać się z funkcjami przejścia neuronu: `hardlim`, `hardlims`, `purelin`, `satlin`, `satlins`, `logsig`, `tansig`, `radbas`. Wykorzystując definicje funkcji zawarte w dodatku I narysować wykresy funkcji przejścia neuronów dla łącznego pobudzenia z przedziału od -10 do 10.
   W sprawozdaniu zamieścić ich wykresy i programy rysujące.



3. Dla łącznego pobudzenia neuronu z p. 1 wyznaczyć wartości wyjścia neuronów o funkcjach przejścia z p. 2.
   hardlim(3.3)
   Out[]: 1
   hardlims(3.3)
   Out[]: 1.0
   purelin(3.3)
   Out[]: 3.3
   satlin(3.3)
   Out[]: 1
   satlins(3.3)
   Out[]: 1
   logsig(3.3)
   Out[]: 0.9644288107273639
   tansig(3.3)
   Out[]: 0.9972829600991421
   radbas(3.3,1)
   Out[]: 1.864374233151685e-05

Sprawozdanie

Na ocenę dobrą opisać szczegółowo poszczególne punkty ćwiczenia wraz z krótkim uzasadnieniem teoretycznym. Na ocenę bardzo dobrą zaproponować własną implementację modelu neuronu o funkcjach przejścia z p.2 przebiegu laboratorium.

Literatura

[1] McCulloch W. S., Pitts W., *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics 5, 115-133.

Dodatki

Dodatek I: Zawartość pliku neuron_def.py

```
import numpy as np
import matplotlib.pyplot as plt

def hardlim(n):
# Funkcja perceptronowa y = 1 dla n>0 i 0 w przeciwnym przypadku
    if np.isscalar(n):
        if n<0:
            return 0
        elif n>=0:
            return 1
        else:
            return n
    else:
        y=np.copy(n)
        y[n<0]=0
        y[n>=0]=1
        return y

def hardlims(n, *b):
# Funkcja perceptronowa symetryczna y = 1 dla n>=0 i -1 w przeciwnym przypadku
    if not b:
        return np.sign(n)
    else:
        return np.sign(np.array(n) + b)[0][0]

def purelin(n, *b):
# Funkcja liniowa y = n
    if not b:
        return n
    else:
        return (n + b * np.ones((np.size(n),1)))[0][0]

def satlin(n):
# Funkcja liniowa ograniczona y = n dla 0<n<=1 i 0 dla n<0 oraz 1 dla n>1
    if np.isscalar(n):
        if n<0:
            return 0
        elif n>1:
            return 1
        else:
            return n
    else:
        y=np.copy(n)
        y[n<0]=0
        y[n>1]=1
        return y

def satlins(n):
# Funkcja liniowa ograniczona y = n dla 0<n<=1 i 0 dla n<0 oraz 1 dla n>1
    if np.isscalar(n):
        if n<-1:
            return -1
        elif n>1:
            return 1
        else:
            return n
    else:
        y=np.copy(n)
        y[n<-1]=-1
```

```
        y[n>1]=1
        return y

def logsig(n, *b):
# Sigmoidalna funkcja unipolarna
    if not b:
        return 1 / (1 + np.exp(-n))
    else:
        return 1 / (1 + np.exp(-(np.array(n) + b)))[0][0]

def tansig(n, *b):
# Sigmoidalna funkcja bipolarna
    if not b:
        return np.tanh(np.array(n))
    else:
        return np.tanh(np.array(n)+b)

def radbas(n,beta=1,*b):
# Funkcja radialna
    if not b:
        return np.exp(-np.power(beta*n,2))
    else:
        return np.exp(-np.power(beta*(n+b),2))


nn_identifiers = ["hardlim", "hardlims", "satlin", "satlins",
                  "purelin", "logsig", "tansig", "radbas"]

def plot_threshold_fuctions(f):
    n = np.linspace(-10,10,1000)
    for f_ind in range(len(f)):
        y = globals()[f[f_ind]](n)
        plt.figure()
        plt.plot(n,y)
        plt.xlabel("n", fontsize = 16)
        plt.title(f[f_ind], fontsize = 20)
        plt.grid()
        plt.draw()

plot_threshold_fuctions(nn_identifiers)
```