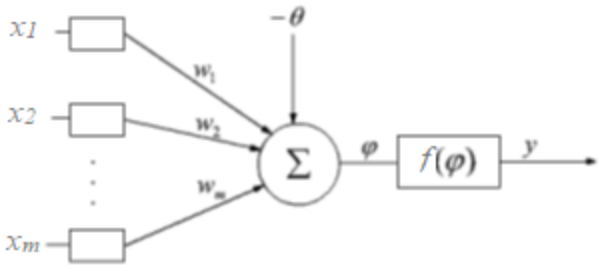
Perceptron

Wprowadzenie – ogólny opis perceptronu

Sztuczne siec neuronowe są obecnie intensywnie rozwijającą się dyscypliną nauki i znajdują bardzo szerokie zastosowania. Sieci oparte na modelu perceptronu są jedne z najbardziej popularnych. Tego typu sieci zawierają neurony nieliniowe, tzn. zastosowana w neuronach funkcja aktywacji jest nieliniowa.



Rysunek 1. Model neuronu typu perceptron prosty

Łączne pobudzenie perceptronu φ może być wyrażone za pomocą wzoru:

$$\varphi = \sum_{i=1}^{m} w_i \cdot x_i - \theta$$

gdzie mamy: θ – wartość progowa, w_i – składowa wektora wag, x_i – składowa wektora wejściowego. Przy założeniu, że $x_0 = -1$ oraz że $w_0 = \theta$ łączne pobudzenie neuronu jest zapisane w następujący sposób:

$$\varphi = \sum_{i=0}^{m} w_i \cdot x_i$$

Funkcja aktywacji w *perceptronie prostym* może być unipolarna:

$$f(\varphi) = \begin{cases} 1 & \varphi \geq 0 \\ 0 & \varphi < 0 \end{cases}$$

lub bipolarna:

$$f(\varphi) = \begin{cases} +1 & \varphi \geq 0 \\ -1 & \varphi < 0 \end{cases}$$

Uczenia tego typu neuronu polega na korekcie wag zgodnie ze wzorem:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \mu\delta^k\mathbf{x}^k$$

gdzie:

$\mathbf{w}^k$ – oznacza wektor wag neuronu dla *k*-tego kroku uczenia,

μ – współczynnik uczenia (zwykle z przedziału od 0 do 1),

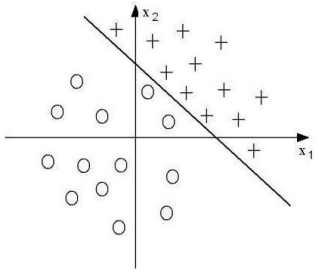
δ^k – błąd dla *k*-tego kroku uczenia, tzn.:

$$\delta^k = f(\varphi)^k - d^k$$

d^k – wartość żądana w *k*-tym kroku uczenia,

$\mathbf{x}^k$ – wektor wejściowy w *k*-tym kroku uczenia.

Proces uczenia odbywa się cyklicznie, aż do momentu, gdy błąd δ^k wszystkich podanych wzorów uczących będzie równy 0. Po zakończeniu uczenia, poprawność otrzymanych wartości wag **w** neuronu można sprawdzić, wykreślając tzw. *linię decyzyjną*.



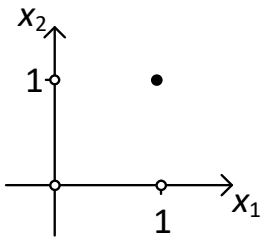
Równanie linii decyzyjnej ma postać:

$$\sum_{i=0}^{m} w_i \cdot x_i = 0$$

Przygotowanie danych

Zadanie polega na nauczeniu perceptronu rozpoznawania zbiorów punktów jako jedynek lub zer. Dodatkowo wykreślona zostanie linia decyzyjna, którą perceptron oddziela od siebie dwa zbiory danych.

Zadaniem perceptronu będzie nauczenie rozpoznawania czterech punktów jak na rysunku poniżej:



Każdy punkt przynależy do jednej z klas: zer (puste kółko) lub jedynek (kółko zamalowane). Zatem, jeżeli na wejście perceptronu podane będzie x_1=0 i x_2=0, wówczas odpowiedzią perceptronu powinno być 0, dla 1 i 0 również odpowiedzią powinno być 0, ale dla pary 1 i 1 podanej na wejście, odpowiedzią powinno być 1. Współrzędne punktów x_1, x_2 oraz klasę *d* można przedstawić w tabeli:

x_1	x_2	*d*
0	0	0
0	1	0
1	0	0
1	1	1

gdzie x_1 i x_2 to współrzędne punktu, *d* to klasa do której należy punkt (biała-czarna, oznaczona jako 0 lub 1). Dane można zatem przechowywać w dwóch tabelach:

```
x = np.array( [ [0,0],
                [0,1],
                [1,0],
                [1,1]])
d = np.array([0,
              0,
              0,
              1])
```

przy czym *np* to biblioteka *numpy*, którą należy zaimportwać.

Perceptron

Kolejną rzeczą jest przygotowanie kodu perceptronu. Aby obliczyć wyjście perceptronu *y*, należy obliczyć wcześniej pobudzenie perceptronu zgodnie ze wzorem:

$$\varphi = \sum_{i=0}^{m} w_i \cdot x_i$$

Perceptron powinien mieć wcześniej ustalone wartości wag. Dla omawianego przypadku, gdzie *m*=2 (przestrzeń dwuwymiarowa), perceptron powinien mieć 3 wagi: w_0, w_1, w_2. Początkowo powinny być one najlepiej dobrane losowo:

```
w = np.random.random(3)
```

Obliczenie φ jest proste i można to obliczyć zwykła sumą:

$$\varphi = w_1 \cdot x_1 + w_2 \cdot x_2 + w_0 \cdot x_0$$

Należy zaznaczyć też, że wartość x_0 jest zawsze równa -1, więc:

$$\varphi = w_1 \cdot x_1 + w_2 \cdot x_2 + w_0 \cdot (-1)$$

Odpowiedź perceptronu *y* jest wartością funkcji unipolarnej, zgodnie z podanym wcześniej wzorem:

$$y = f(\varphi) = \begin{cases} 1 & \varphi \geq 0 \\ 0 & \varphi < 0 \end{cases}$$

Należy sprawić dla każdej pary x_1, x_2 czy odpowiedź perceptrony *y* jest taka sama jak wartość oczekiwana *d*. W Pythonie program ten będzie wyglądał następująco:

```
for i in range( len(x) ):
  xx = x[i]  #pobranie kolejnych wspolrzednych punktu xx[0] to x1, xx[1] to x2 z tabeli
  dd = d[i]  #dd to klasa do której przynależy punkt xx

  s = xx[0]*w[1] + xx[1]*w[2] + w[0]*(-1)
  if s >= 0:
    y = 1
  else:
    y = 0

  print("x1:",xx[0], "x2:",xx[1], " d:",dd, "  y:",y)
```

Dla każdego punktu i obliczana jest pobudzenie neuronu oznaczone jako s, następnie funkcja unipolarna w zależności od wartości pobudzenia przyjmuje wartość 0 lub jeden, która jest odpowiedzią neuronu. Uruchamiając kilkakrotnie kod dla różnych losowych wartości wag można zauważyć, że odpowiedzi perceptronu różnią się od oczekiwanych wartości d.

```
x1: 0 x2: 0  d: 0   y: 1
x1: 0 x2: 1  d: 0   y: 0
x1: 1 x2: 0  d: 0   y: 0
x1: 1 x2: 1  d: 1   y: 0
```

Uczenie perceptronu

Aby perceptron poprawnie rozpoznawał dane wartości wejściowe i generował odpowiedź zgodną z oczekiwaną, należy dostroić wagi. Wagi dostraja się w procesie nauczania zgodnie ze wzorem przedstawionym we wprowadzeniu:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \mu\delta^k\mathbf{x}^k$$

Nauczanie jest odbywa w kolejnych krokach (zwanych iteracjami), oznaczonymi jako *k*. Waga zmienia się w kolejnych krokach w zależności od błędu δ oraz wartości wejściowej $\mathbf{x}$. Błąd δ to różnica pomiędzy wartością oczekiwaną *d* a wartością, którą zwrócił perceptron *y* (czyli $f(\varphi)$):

$$\delta = y - d.$$

Proszę zwrócić uwagę, że jeżeli $\delta = 0$, a tak jest w przypadku, gdy perceptron odpowiedział poprawnie, wówczas wagi w nie zmieniają się w kolejnym kroku.

Parametr μ, czyli współczynnik uczenia, przyjmuje najczęściej wartość 0,1. Mając te informacje, wagi dla wszystkich próbek x_1 i x_2 wraz z ich oczekiwanymi wartościami *d* można zmieniać na przykład w taki sposób:

```
mi = 0.1
for i in range(len(x)):
  k+=1
  xx = x[i]  #pobranie kolejnych wspolrzednych punktu xx[0] to x1, xx[1] to x2 z tabeli
  dd = d[i]  #dd to klasa do której przynależy punkt xx

  s = xx[0]*w[1] + xx[1]*w[2] + w[0]*(-1)
  if s >= 0:
    y = 1
  else:
    y = 0

  w[0] = w[0] + mi*(dd-y)*(-1)     #x0=-1
  w[1] = w[1] + mi*(dd-y)*(xx[0])  #xx[0] to x1
  w[2] = w[2] + mi*(dd-y)*(xx[1])  #xx[1] to x2
```

W ten sposób wykonano cztery modyfikacje wag (tyle jest punktów w wektorze *x*), czyli *k*=4. Ponowne sprawdzenie poprawności odpowiedzi perceptronu poprzednim kodem wciąż nie przynosi oczekiwanego rezultatu:

```
x1: 0 x2: 0  d: 0   y: 1
x1: 0 x2: 1  d: 0   y: 1
x1: 1 x2: 0  d: 0   y: 0
x1: 1 x2: 1  d: 1   y: 0
```

Dopiero kilkunastokrotne wykonanie operacji uczenia daje poprawne wyniki:

```
x1: 0 x2: 0  d: 0   y: 0
x1: 0 x2: 1  d: 0   y: 0
x1: 1 x2: 0  d: 0   y: 0
x1: 1 x2: 1  d: 1   y: 1
```

Linia decyzyjna

Linia decyzyjna pozwala perceptronowi na podział płaszczyzny na dwie części: część, gdzie wszystkie punkty będą oznaczone jako zero, a w pozostałej punkty będą oznaczone jako jedynki. We wprowadzeniu podano wzór:

$$\sum_{i=0}^{m} w_i \cdot x_i = 0$$

Dla rozpatrywanego przypadku, gdy *m*=2, wzór przyjmuje postać:

$$w_1 \cdot x_1 + w_2 \cdot x_2 + w_0 \cdot x_0 = 0$$

Wzór na linię, znany z matematyki, to

$$y = a \cdot x + b$$

zatem przedstawiony wcześniej wzór należy przekształcić do takiej postaci. Na wykresie z punktami, oś Y oznaczona jest jako x_2, natomiast oś X jako x_1. Zatem można wyprowadzić wzór na x_2 w następujący sposób:

$$w_2 \cdot x_2 = -w_1 \cdot x_1 - w_0 \cdot x_0$$

następnie, dzieląc obustronnie przez w_2 i przyjmując, że x_0=1 otrzymujemy wzór na prostą w postaci:

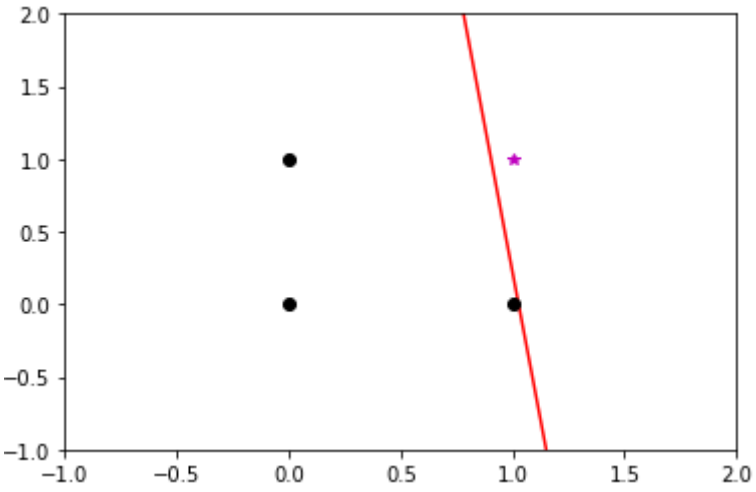
$$x_2 = -\frac{w_1}{w_2} \cdot x_1 + \frac{w_0}{w_2}$$

Otrzymujemy zatem równanie prostej, gdzie $a = -\frac{w_1}{w_2}$ i $b = -\frac{w_0}{w_2}$.

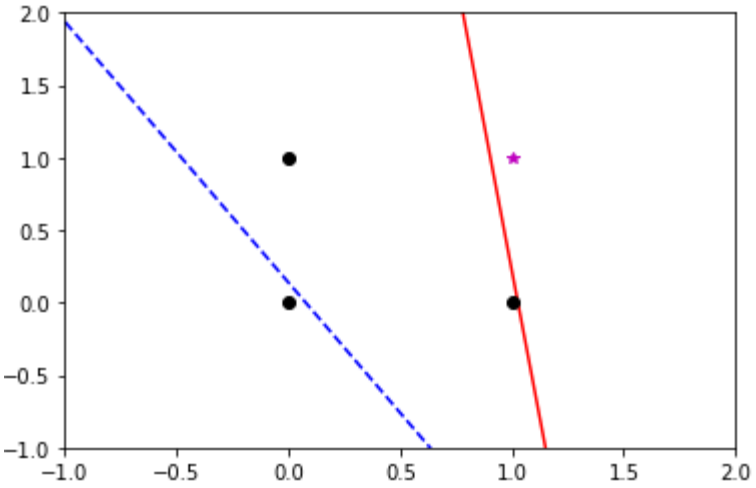
Korzystając z biblioteki matplotlib można zatem na jednym wykresie narysować punkty oraz wykres prostej przy pomocy poniższego kodu

```
1 import matplotlib.pyplot as plt
2 xx = np.arange(-1,3)
3 yy = -(w[1]/w[2]) * xx + (w[0]/w[2])
4 plt.plot(xx, yy, 'r-')
5
6 for i in range(len(x)):
7   if d[i] == 0:
8     plt.plot(x[i,0], x[i,1], 'ko')
9   else:
10    plt.plot(x[i,0], x[i,1], 'm*')
```

Uzyskany wykres prezentuje się jak na rysunku poniżej. Widać wyraźnie, że linia odcina punkty jedynki (oznaczone gwiazdką) od punktów zer (oznaczonych kółkiem).



A tak wygląda porównanie z linią decyzyjną uzyskaną przed procesem uczenia (niebieska przerywana):



Zadanie

1. Korzystając z kratki papieru w kratkę, należy wygenerować (narysować) zbiór 40 dowolnych punktów. Należy nanieść osie X i Y tak, aby można było odczytywać współrzędne punktów. Następnie zbiór punktów należy przedzielić linią, dzieląc punkty na dwie mniej więcej równe połowy. Punkty nad linią będą należeć do kategorii jedynek, punkty pod linią do kategorii zer. Wprowadź punkty do odpowiednich tabel w Pythonie.
2. Należy graficznie pokazać wygenerowany zbiór, rozróżniając przynależność elementów zbioru do poszczególnych klas na przykład różnym kolorem punktów.
3. Należy wykonać poniższe punkty:
    a. Przygotować perceptron prosty.
    b. Narysować linię decyzyjną nienauczonego perceptronu naniesioną na wykres z wygenerowanymi punktami (podobny do wykresu uzyskanego w punkcie 2). Należy odróżnić poprawnie i niepoprawnie sklasyfikowane przez perceptron próbki.
    c. Przeprowadzić uczenie neuronu, korzystając z utworzonego zbioru uczącego. Ustaw maksymalną liczbę iteracji uczenia całego zbioru danych na 100.
    d. Narysuj kolejny wykres z punktami. Narysuj linię decyzyjną perceptronu po przeprowadzonym uczeniu. Zaznacz punkty poprawnie i niepoprawnie sklasyfikowane. Należy dorysować linię decyzyjną perceptronu sprzed procesu uczenia (punkt b).
4. Dodatkowe zadania:
    a. Zatrzymaj uczenie perceptronu wtedy, gdy perceptron będzie nauczony, czyli gdy liczba błędów uzyskanych dla całego kompletu próbek wyniesie zero. Podaj liczbę iteracji.
    b. Sprawdzić działanie perceptronu podając 3 nowe, dowolne punkty (np. wprowadzone współrzędne z klawiatury), którymi perceptron nie zostały wcześniej uczony. Sprawdź jak je sklasyfikował perceptron. Umieść te dodatkowe punkty na wykresie z linią decyzyjną.