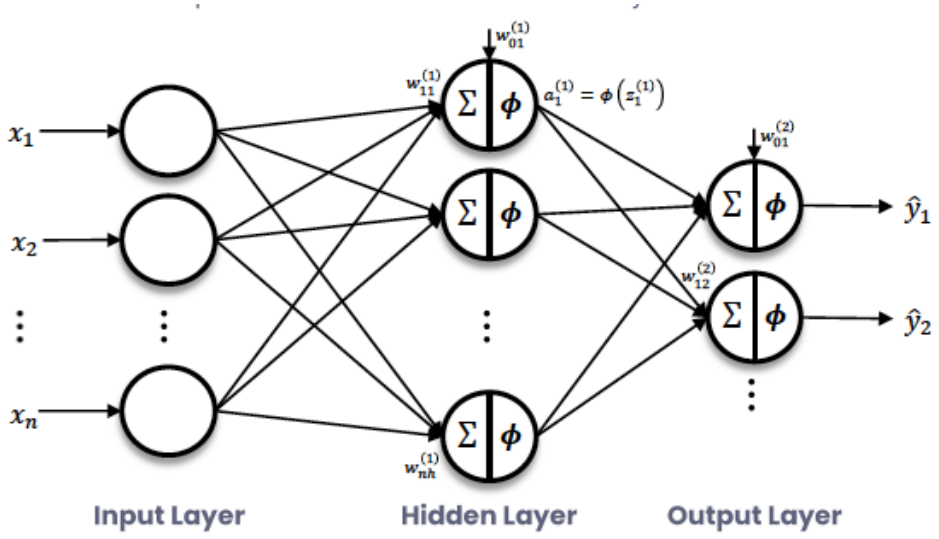
2. Rețele neuronale feedforward

Rețelele cu un singur neuron nu pot rezolva probleme de clasificare liniar neseparabile. Spre exemplu, o rețea cu un singur neuron nu poate rezolva problema învățării funcției booleene XOR. Pentru ca o rețea să poată rezolva probleme de clasificare non-liniare este necesar să avem: 1) o funcție de activare și 2) introducerea mai multor straturi.

Rețele feedforward de neuroni multistrat

Rețelele neuronale feedforward sunt rețele de neuroni grupați pe straturi, în care propagarea informației se realizează numai dinspre intrare spre ieșire (de la stânga la dreapta). În figura de mai jos este reprezentată grafic o *rețea feedforward cu două straturi* (layere). Rețeaua are *n* intrări și deci *n* neuroni pe stratul de intrare (input layer), *h* neuroni pe stratul ascuns (hidden layer) și un număr de neuroni pe stratul de ieșire.



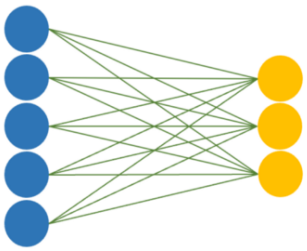
Pentru fiecare perceptron *j* de pe stratul *l* identificăm următoarele:

- ➔ $w_{ij}^{(l)}$ este ponderea neuronului *i* de pe stratul *l* - *1* către neuronul *j* pe stratul *l*. Stratul de intrare este stratul 0. În figura de mai sus sunt exemplificate ponderile $w_{11}^{(1)}$, $w_{nh}^{(1)}$, $w_{12}^{(2)}$;
- ➔ $w_{0j}^{(l)}$ este bias-ul (deplasarea) neuronului *j* de pe stratul *l*. În figura de mai sus sunt exemplificate bias-urile $w_{01}^{(1)}$, $w_{01}^{(2)}$;;
- ➔ $z_j^{(l)}$ este ieșirea neuronului *j* de pe stratul *l* după însumarea intrărilor ponderate de la toți ceilalți neuroni. În figura de mai sus este exemplificată ieșirea $z_1^{(1)}$;
- ➔ $a_j^{(l)}$ este ieșirea neuronului *j* de pe stratul *l* după aplicarea funcției de activare ϕ ieșirii $z_j^{(l)}$. În figura de mai sus este exemplificată ieșirea $a_1^{(1)}$;

Neuronii de pe primul strat sunt singurii care primesc semnale din exterior (semnalele de intrare). Neuronii dintr-un strat sunt direct conectați cu neuronii din stratul următor; nu există conexiuni între neuronii aceluiași strat. Primul strat se numește ***strat de intrare*** (*input layer*), ultimul strat se numește ***strat de ieșire*** (*output layer*) iar celelalte straturi se numesc ***straturi ascunse*** (*hidden layers*).

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \end{bmatrix} \cdot \begin{bmatrix} i_0 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

$$= \begin{bmatrix} w_{0,0} \cdot i_0 + w_{0,1} \cdot i_1 + w_{0,2} \cdot i_2 + w_{0,3} \cdot i_3 + w_{0,4} \cdot i_4 + b_0 \\ w_{1,0} \cdot i_0 + w_{1,1} \cdot i_1 + w_{1,2} \cdot i_2 + w_{1,3} \cdot i_3 + w_{1,4} \cdot i_4 + b_1 \\ w_{2,0} \cdot i_0 + w_{2,1} \cdot i_1 + w_{2,2} \cdot i_2 + w_{2,3} \cdot i_3 + w_{2,4} \cdot i_4 + b_2 \end{bmatrix}$$



Rezolvarea problemei XOR folosind o rețea feedforward multistrat

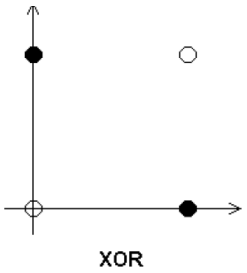
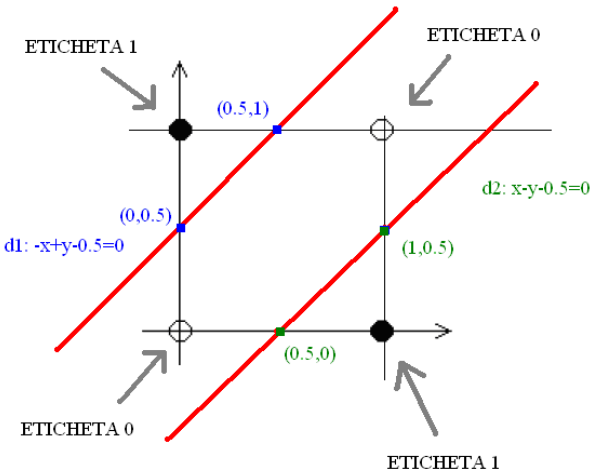


Figura alăturată conține reprezentarea grafică a funcției XOR, în care punctele (0,0), (1,1) au eticheta 0 (alb), iar punctele (1,0), (0,1) au eticheta 1 (negru). O rețea cu un singur neuron nu poate rezolva problema învățării funcției booleene XOR. Acest obstacol poate fi depășit folosind o rețea cu mai mulți perceptroni și mai multe straturi.

Pentru problema XOR, limitarea perceptronului poate fi depășită prin crearea unei rețele de neuroni cu 2 straturi cu următoarea structură:

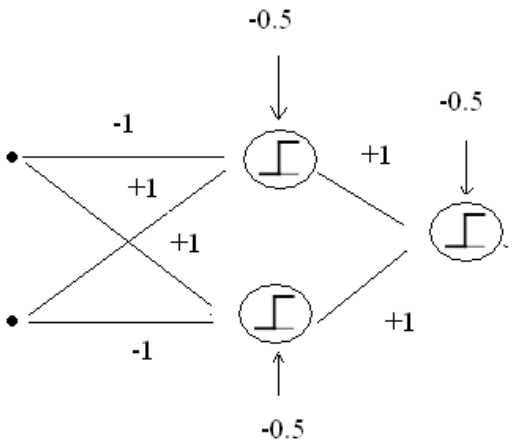
- primul strat conține doi neuroni, fiecare neuron implementează ecuația unei drepte (culoare roșie în figura de mai jos) care separă un punct negru de celelalte două puncte albe. Fiecare neuron stabilește dacă punctul (x,y) se află deasupra sau dedesubtul celor două drepte.
- al doilea strat conține un neuron care stabilește dacă punctul (x,y) se află între cele două drepte, ieșirea rețelei fiind astfel 0 sau 1, în caz contrar.



Există o infinitate de drepte roșii care separă un punct negru de celelalte două puncte albe. Particularizând dreptele de culoare roșie ca în figura alăturată obținem dreptele de separare:

d_1: -x + y -0.5 = 0
d_2: x -y - 0.5 = 0.

Punctele (x,y) deasupra dreptei d_1 au valoarea ieșirii neuronului de pe primul strat corespunzător 1, iar cele de dedesubtul dreptei d_1 au valoarea 0. Punctele deasupra dreptei d_2 au valoarea ieșirii neuronului de pe primul strat corespunzător 0, iar cele de dedesubtul dreptei d_2 au valoarea 1. În final, punctul (x,y) este etichetat cu 0 dacă se află între cele două drepte și cu 1 altfel. Rețeaua pe care o putem construi poate fi reprezentată grafic ca în figura alăturată, unde funcțille de activare ale fiecărui neuron sunt funcțiile hardlim:

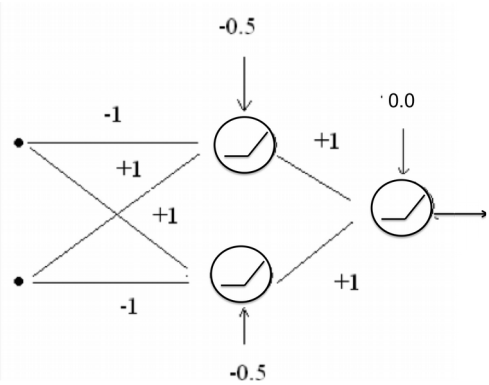


$$\text{hardlim}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Pentru exemplul de mai sus al funcției XOR ne-am putut da seama foarte ușor care este o arhitectură de rețea care rezolvă problema. În practică, pentru probleme mai grele acest lucru este aproape imposibil de realizat. Soluția este învațarea ponderilor unei rețele din datele de antrenare. Rețelele multistrat feedforward se antrenează folosind algoritmul backpropagation (propagare înapoi a erorii), algoritm care va fi prezentat la curs. Conform acestui algoritm actualizarea ponderilor se face utilizând derivatele unei funcții de eroare care încearcă să minimizeze diferența dintre rezultatul rețelei și etichetele dorite. În calculul derivatei funcției eroare intervin și derivatele funcțiilor de activare. Funcția *hardli*m este discontinuă în punctul x = 0, iar în toate celelalte puncte derivata ta este 0. Ea nu poate fi folosită la învățare. Pentru antrenare se folosesc alte funcții de activare.

Funcțiile standard de activare din biblioteca scikit-learn sunt:

1. funcția identitate $f(x) = x$;
2. funcția logistică $f(x) = 1/(1+e^{-x})$;
3. funcția tangentă hiperbolică $f(x) = tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$;
4. funcția relu $f(x) = max(0, x)$.



Rețeaua de mai sus folosită pentru rezolvarea problemei XOR poate fi convertită la o altă rețea în Python folosind din biblioteca scikit-learn clasa ***MLPClassifier*** pentru definitea unei rețele de neuroni (perceptroni) cu funcția de transfer „relu" și asignarea manuală a ponderilor, confirm desenului alăturat. Codul Python este următorul:

```
X = np.array([0, 0, 0, 1, 1, 0, 1, 1]).reshape(4, 2)
y = np.array([0, 1, 1, 0]).reshape(4,)
mlp = MLPClassifier(hidden_layer_sizes=(2,), activation='relu')
mlp.fit(X, y)
mlp.coefs_ = [np.array([[ -1.0 ,  1.0], [1.0, -1.0]]), np.array([[1.0],[1.0]])]
mlp.intercepts_ = [np.array([-0.5, -0.5]), np.array([0.0])]
y_pred=mlp.predict(X)                    # prediction
print(y_pred)
```

Definirea unei retele de perceptroni in Scikit-learn

```
from sklearn.neural_network import MLPClassifier # importul clasei

mlp_classifier_model = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam',
alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5,
max_iter=200, shuffle=True, random_state=None, tol=0.0001, momentum=0.9, early_stopping=False,
validation_fraction=0.1, n_iter_no_change=10)
```

Parametrii modelului sunt:

1. *hidden_layer_sizes* (***tuple, lungime = n_layers - 2, default=(100,)***): al *i-lea* element reprezintă numărul de neuroni din al *i-lea* strat ascuns.
2. *activation* ({**'identity', 'logistic', 'tanh', 'relu'**}, **default='relu') –** funcțiile de activare
   a. 'Identity' (funcția identitate): $f(x) = x$
   b. 'logistic' (funcția logistică): $f(x) = 1/(1+e^{-x})$
   c. 'tanh' (funcția tangentă hiperbolică): $f(x) = tanh(x)$
   d. 'relu' (funcția liniară rectificată): $f(x) = max(0, x)$
3. *solver* ({**'lbfgs', 'sgd', 'adam'}, default='adam')**: algoritmul folosit pentru învățarea ponderilor (actualizarea ponderilor)

4. *batch_size*: (***int, default='auto'***), auto - marimea batch-ului pentru antrenare este *min(200, n_samples)*.
5. *learning_rate_init* (**double, default=0.001**): rata de învățare
6. *max_iter* (**int, default=200**): numărul maxim de epoci pentru antrenare
7. *shuffle* (**bool, default=True**): amesteca datele la fiecare epocă
8. *tol* (**float, default=1e-4**) : daca eroarea sau scorul nu se îmbunătățesc timp de *n_iter_no_change* epoci consecutive (*și learning_rate != 'adaptive'*) cu cel puțin *tol*, antrenarea se oprește.
9. *n_iter_no_change* **: (int, optional, default 10,** *sklearn-versiune-0.20*): numărul maxim admis de epoci fara scăderea erorii.
10. *alpha* **(float, default=0.0001)**: parametru pentru regularizarea L_2.
11. *learning_rate* ({**'constant', 'invscaling', 'adaptive'**},**default='constant'**):
    a. **'constant'** : rata de învățare este constantă și este dată de parametrul *learning_rate_init*.
    b. **'invscaling'**: rata de învățare va fi scăzută la fiecare pas *t*, după formula: new_learning_rate = learning_rate_init / pow(t, power_t)
    c. **'adaptive'**: păstreaza rata de învățare constantă cât timp eroarea scade. Dacă eroarea nu scade cu cel puțin *tol* (față de epoca anterioară) sau dacă scorul pe mulțimea de validare (*doar dacă ealy_stopping=True)* nu crește cu cel putin *tol* (față de epoca anterioară), rata de învățare curentă se împarte la 5.
12. power_t (**double, default=0.5**): parametrul pentru learning_rate='invscaling'.
13. momentum **(float, default=0.9):** valoarea pentru momentum cand se foloseste gradient descent cu momentum. Trebuie sa fie între 0 si 1.
14. early_stopping (**bool, default=False**): dacă este setat cu *True* atunci antrenarea se va termina dacă eroarea pe mulțimea de validare nu se îmbunătățește timp de *n_iter_no_chage* epoci consecutive cu cel puțin *tol*.
15. validation_fraction **(float, optional, default=0.1)**: procentul din mulțimea de antrenare care să fie folosit pentru validare (doar cand *early_stopping=True*). Trebuie să fie între 0 și 1.

Atribute:

16. **classes_** : array sau o lista de array de dimensiune (n_classes,):
    a. clasele pentru care a fost antrenat clasificatorul.
17. **loss_** : float, eroarea actuala
18. **coefs_** : lista, lungimea = n_layers – 1
    a. al ***i***-lea element din lista reprezinta matricea de ponderi dintre stratul ***i*** si ***i + 1***.
19. **intercepts_ :** lista, lungimean_layers – 1
    a. al ***i***-lea element din lista reprezinta vectorul de bias corespunzator stratului ***i + 1***.
20. **n_iter_** : int, numarul de epoci parcurse pana la convergenta.
21. **n_layers_** : int, numarul de straturi.
22. **n_outputs_** : int, numarul de neuroni de pe stratul de iesire.
23. **out_activation_** : string, numele functiei de activare de pe stratul de iesire.

Funcții:

1. **mlp_classifier_model.fit(X, y)**:
    a. antreneaza modelul pe datele de antrenare X si etichetele y cu parametrii setati la declarare.
    b. X este o matrice de dimensiune (n_samples, n_features).
    c. y este un vector sau o matrice de dimensiune (n_samples,) - pentru clasificare binara si regresie, (n_samples, n_outputs) pentru clasificare multiclass.
    d. returneaza modelul antrenat.

2. **mlp_classifier_model.predict(X):**
   a. prezice etichetele pentru X folosind ponderile invatate.
   b. X este o matrice de dimensiune (n_samples, n_features).
   c. returneaza clasele prezise intr-o matrice de dimensiune
      (n_samples,)- pentru clasificare binara si regresie, (n_samples, n_outputs) pentru clasificare multiclass.

3. **mlp_classifier_model.predict_proba(X):**
   a. prezice probabilitatea pentru fiecare clasa.
   b. X este o matrice de dimensiune (n_samples, n_features).
   c. returneaza o matrice de (n_samples, n_classes) avand pentru fiecare exemplu si pentru fiecare clasa probabilitatea ca exemplul sa se afle in clasa respectiva.

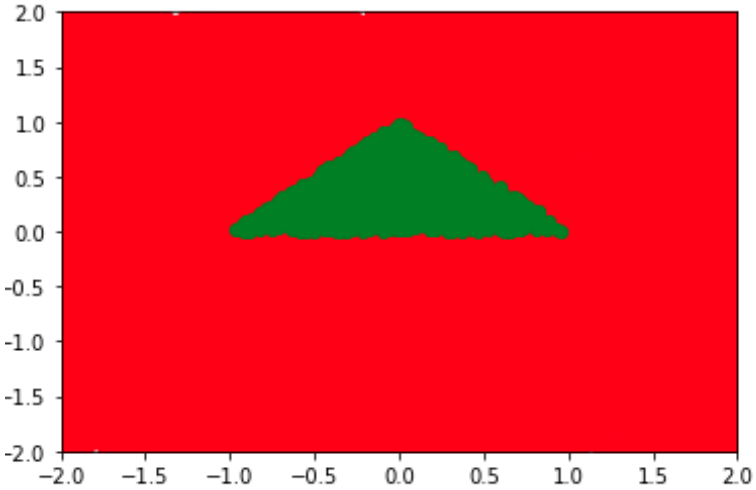
4. **mlp_classifier_model.score(X, y):**
   a. returneaza acurateta medie in functie de X si y.
   b. X este o matrice de dimensiune (n_samples, n_features).

y are dimeniunea (n_samples,) - pentru clasificare binara si regresie, (n_samples, n_outputs) pentru clasificare multiclas

Exerciţii

a) Scrieți o funcție care să implementeze funcția indicator a triunghiului ABC de vârfuri A(-1,0), B(0,1), C(1,0). Funcția indicator ia valoarea 1 pentru punctele din interiorul triunghiului şi de pe frontieră şi 0 în rest.

b) Construiți matricea **Puncte** de dimensiune 20000×2 care să conțină 20000 de puncte generate uniform în pătratul [-2 2] × [-2 2]. Aplicați funcția indicator pe datele **Puncte** şi plotați punctele colorându-le diferit în funcție de răspunsul funcției: culoarea roşie pentru punctele din afara triunghiului şi culoarea verde pentru punctele din interiorul triunghiului. Ar trebuie să obțineți o figură asemănătoare cu cea de mai jos:



c) Antrenați o rețea pe primele 10000 de puncte din matricea **Puncte** care să învețe funcția indicator a triunghiului ABC. Plotați rezultatele obținut pe celelalte puncte (cele 10000 rămase) colorându-le diferit în funcție de răspunsul rețelei: culoarea roşie pentru punctele din afara triunghiului şi culoarea verde pentru punctele din interiorul triunghiului.

d) Care este performanța rețelei pe cele 10.000 de puncte testate?

e) Comparați MLP şi perceptronul pe datele de antrenare ale proiectului.