Sieci Neuronowe

Zadanie 1

Napisz wywołanie funkcji `perceptron`, która będzie realizowała obliczenia perceptronu z dwoma wejściami x_1 oraz x_2, z progową funkcją aktywacji. Jako wejście będzie przyjmowała wagi w_1, w_2 oraz wejścia $x_{1,}x_2 \in \{0, 1\}$ Dobierz w sposób manualny wagi w1, w2 oraz w3 do problemu:

```
def perceptron(w1, w2, w3, x1, x2):
    y = w1 * x1 + w2 * x2 + w3
    return 1 if y > 0 else 0
```

1. Operacji logicznej AND:

   Przykład: parametryzacja postaci w1=0.2, w2=0,w3=0.1 nie jest dobra mimo, że dla x1=1 oraz dla x2=1 wartość $x_1 AND\, x_2$ jest poprawna to już dla x1=0 oraz x2=1 wartość dalej wynosi 1, podczas gdy powinna 0.

x_1	x_2	$x_1 AND\, x_2$
0	0	0
0	1	0
1	0	0
1	1	1

2. Operacji logicznej OR:

x_1	x_2	$x_1 OR\, x_2$
0	0	0
0	1	1

Kamil Żbikowski, kamil.zbikowski@pw.edu.pl

1	0	1
1	1	1

3. Czy podobnie uda się dobrać wagi dla problemu XOR? Odpowiedź uzasadnij.

x_1	x_2	$x_1 XOR\, x_2$
0	0	0
0	1	1
1	0	1
1	1	0

W kolejnych zadaniach Twoim będziemy zajmować się konstruowaniem 3-warstwowej sieci neuronowej. Sieć będziemy uczyć w trybie batchowym. Załóżmy, że przykładowe macierze x (zmienne objaśniające, zbiór treningowy) oraz y (zmienne objaśniane, zbiór treningowy) mają w wierszach przykłady a w kolumnach cechy. Dla przykładu, problem koniunkcji logicznej w postaci macierzy x,y będzie reprezentowany następująco:

```
x
```

```
array([[0, 0],
       [1, 0],
       [0, 1],
       [1, 1]])
```

```
y
```

```
array([[0],
       [0],
       [0],
       [1]])
```

Zadanie 2

Napisz procedurę `train_ann`, która na wyjściu zwróci krotkę zawierającą dwie wartości: w1 oraz w2, które odpowiednio oznaczają macierze wag pomiędzy warstwą wejściową a warstwą ukrytą oraz pomiędzy warstwą ukrytą a warstwą wyjściową. Jest to funkcja, której zadaniem jest trenowanie sieci w trybie batchowym. To o czym należy pamiętać to:

1. Zainicjalizowanie zmiennych odpowiedzialnych za przechowywanie macierzy w1 i w2 w sposób losowy (np. ze standardowego rozkładu normalnego poprzez `np.random.randn`)
2. W zmiennej `it_nmb` przechowujemy maksymalną liczbę iteracji (jest to brutalny warunek stopu nie odwołujący się do wniosków, które należałoby wyciągać wraz z kolejnymi iteracjami, ale dobry “na początek”)
3. Iterując się po od 0 do `it_nmb` należy wykonywać na przemian fazę FeedForward oraz fazę BackPropagation. Pierwsza z nich oblicza wyjście dla zbioru treningowego dla aktualnych wartości macierzy wag w1 oraz w2. Druga aktualizuje te wagi zgodnie z kierunkiem największego spadku.
4. Skorzystaj ze zmiennych pomocniczych aby określić wymiarowość macierzy w1 i w2: `nmb_of_inputs_in_w1, nmb_of_inputs_in_w2, nmb_of_outputs.`
5. Implementacja funkcji compute_feed_forward oraz `compute_backpropagation` będzie przedmiotem kolejnych zadań. Możesz na razie zdefiniować pustą implementację tych funkcji zwracającą “None”.

```
def train_ann(x, y, hidden=10, eta=0.1, it_nmb=10000):
    nmb_of_inputs_in_w1 = x.shape[1] + 1
    nmb_of_inputs_in_w2 = hidden + 1
    nmb_of_outputs = y.shape[1]

    w1 = ...
    w2 = ...

    for i in range(0, it_nmb):
        outputs = compute_feed_forward(x, w1, w2)
        new_w = compute_backpropagation(x, y, outputs[0], outputs[1], w1, w2, eta)
        w1 = new_w[0]
        w2 = new_w[1]
    return (w1, w2)
```

Zadanie 3

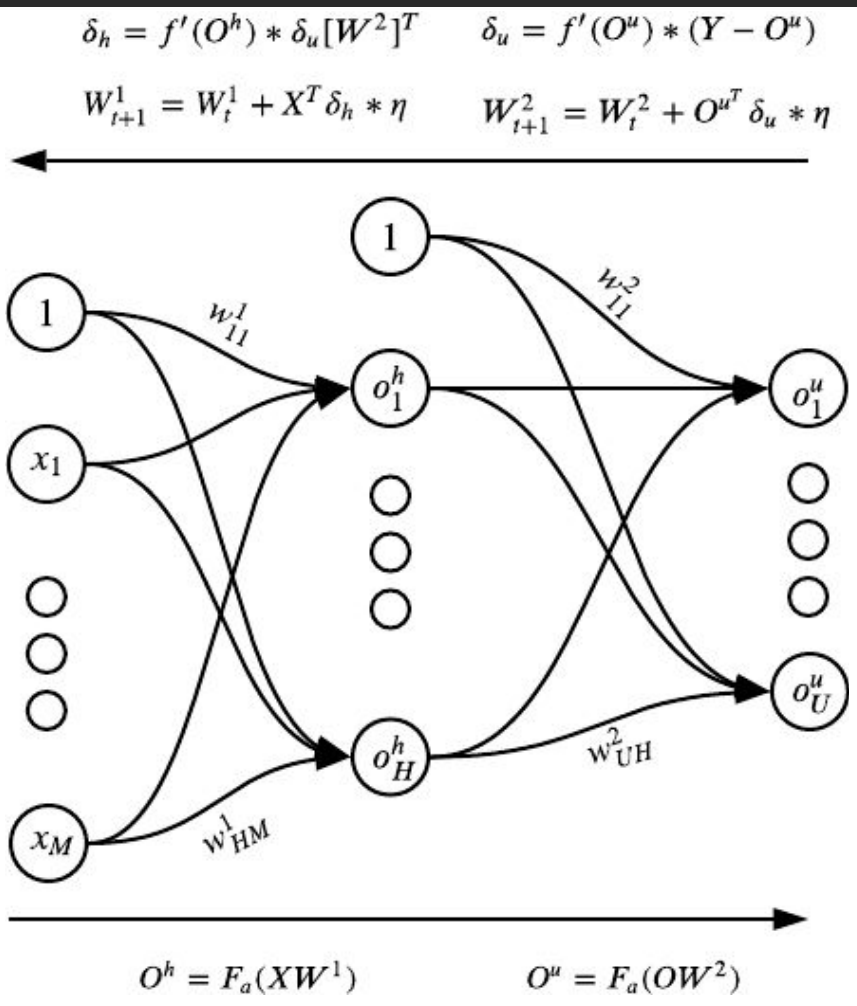
W pierwszym kroku napisz procedurę `compute_feed_forward`, która na wyjściu zwróci listę zawierającą dwie wartości: ou oraz oh, które odpowiednio oznaczają wyjście z warstwy wyjściowej oraz wyjście z warstwy ukrytej.

W zadaniu wykorzystaj implementację funkcji sigmoid.

```
from numpy import exp

def sigmoid(x):
   return 1 / (1 + exp(-x))


def compute_feed_forward(x, w1, w2, activation_function = sigmoid):
   ...
   return (ou, oh)
```



Zadanie 4

Zaimplementuj funkcję `compute_backpropagation` realizującą zadanie wstecznej propagacji.

```
def compute_backpropagation(x, y, ou, oh, w1, w2, eta):
   delta_u = ...
   oh_b = np.c_[oh, np.repeat(1, oh.shape[0])]
   delta_h = ...
   x_b = np.c_[x, np.repeat(1, x.shape[0])]

   w2 = ...
   w1 = ...

   return (w1, w2)
```

Zadanie 5

Zaimplementuj funkcję `predict`, której zadaniem jest dla zadanego zbioru przykładów (którego postać macierzowa jest identyczna jak zmiennej x wykorzystywanej do uczenia) obliczyć wartości zmiennej objaśnianej (czyli macierz/wektor postaci macierzowej identycznej jak zmiennej y wykorzystywanej do uczenia).

```
def predict(x, model, activation_function = sigmoid):
   x = np.c_[x, np.repeat(1, x.shape[0])]
   h = activation_function(x @ model[0])
   h2 = np.c_[h, np.repeat(1, h.shape[0])]
   out = activation_function(...)
   return np.array([1.0 if element >0.5 else 0.0 for element in out])
```

Zadanie 6

Do dzieła! Teraz skoro mamy już w pełni działającą sieć neuronową możemy jej użyć! Najpierw sprawdźmy czy sieć nauczy się rozwiązywać problem koniunkcji logicznej:

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

model = train_ann(x,y)
predict(x, model)
```

Zmień postać macierzy x oraz y tak aby nauczyć się problemu XOR z zadania 2.

Zadanie 7

Podobnie jak w zadaniu z SVM załaduj zbiór “cats”:

```
import pandas as pd

cats = pd.read_csv("cats.csv")
```

a) Zbuduj model dla tego zbioru z wykorzystaniem metod do trenowania sieci neuronowych z poprzedniego zadania. Konieczna będzie transformacja danych wejściowych:

```
x = cats.iloc[:,1:]
y = cats.iloc[:,0]

y.loc[y=="F"] = 1.0
y.loc[y=="M"] = 0.0

y = y.to_numpy().reshape(-1,1).astype(np.float16)
x = x.astype(np.float16)
```

Co zuważyłeś? Czy model, który otrzymałeś wykazuje zdolność uczenia z przedstawionych danych? Do analizy

b) Dokonaj przeskalowania danych zgodnie z poniższą formułą:

```
from sklearn import preprocessing

x_scaled = preprocessing.scale(x)
```

Co zaobserwowałeś? Jeśli skalowanie pomogło - to wyjaśnij dlaczego?

Porównaj wyniki otrzymane z wynikami modelu SVM. Do rysowania krzywej decyzyjnej użyj zmodyfikowanej wersji funkcji plotCats z poprzedniego laboratorium:

```
import matplotlib.pyplot as plt


def plotCats(X, Y, model):
   x_min, x_max = X[:,0].min() - .5, X[:,0].max() + .5
   y_min, y_max = X[:,1].min() - .5, X[:,1].max() + .5
   h = .02
   xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
   predictions = predict(np.c_[xx.ravel(), yy.ravel()], model)

   predictions = predictions.reshape(xx.shape)

   f, ax = plt.subplots(figsize=(8, 6))
   contour = ax.contourf(xx, yy, predictions, 1, cmap="RdBu",
                         vmin=0, vmax=1)
```

```
    ax_c = f.colorbar(contour)
    ax_c.set_label("Decision class")

    ax.scatter(X[:,0], X[:,1], c=y, s=55,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="black", linewidth=1)

    ax.set(
       xlim=(x_min, x_max), ylim=(y_min, y_max),
       xlabel="Bwt", ylabel="Hwt")
```

c) Zastosuj skalowanie Min - Max i porównaj wyniki.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(x)
x_minmax_scaled = scaler.transform(x)
x_minmax_scaled.std(axis=0)
```