GŁĘBOKIE UCZENIE – LABORATORIUM 2024

Temat ćwiczeń: ***Wprowadzenie do sieci neuronowych. Prognozowanie neuronowe: propagacja w przód (elementarz)***

Prowadzący: Piotr Pięta, pieta@agh.edu.pl

Literatura przedmiotu:

[1] http://neuralnetworksanddeeplearning.com/

[2] http://www.deeplearningbook.org/

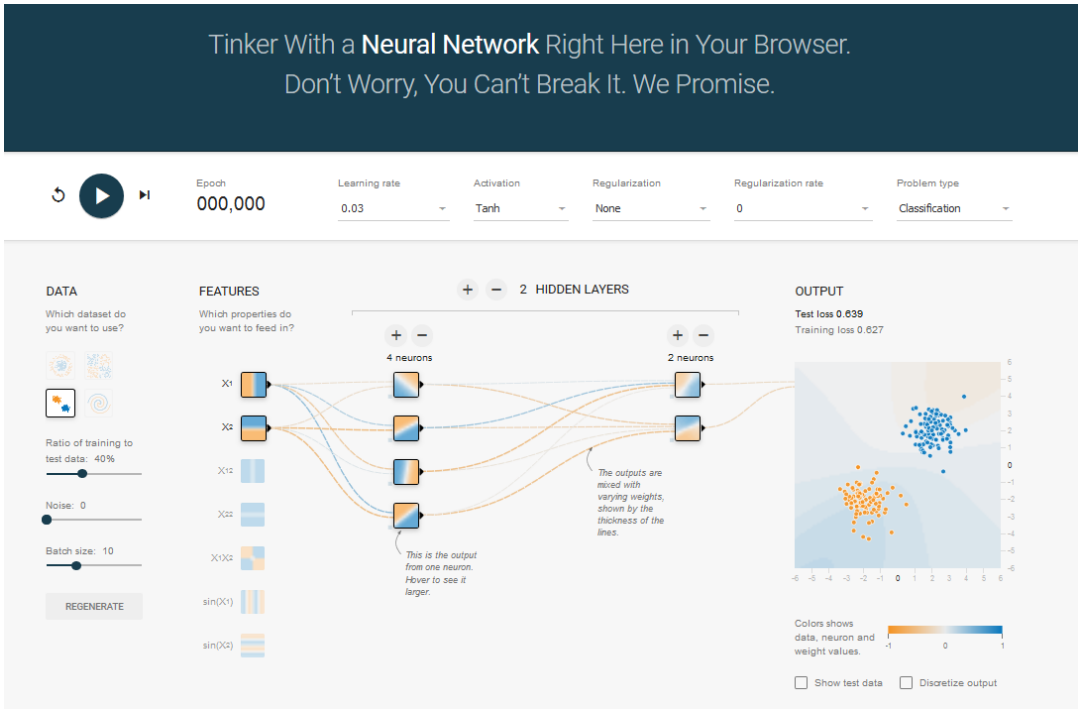
[3] **http://cs231n.github.io/python-numpy-tutorial/**

Przypominam, sugerowanym środowiskiem pracy programistycznej jest notebook *Jupyter* (nic nie stoi na przeszkodzie, aby wykorzystać inne narzędzie, w szczególności środowisko *Anaconda*:

https://docs.anaconda.com/anaconda/install/)

Instrukcja (proszę Państwa o zwarte odpowiedzi) ☺ TODO

0. Znajdź informację (*dowolne źródło*) i napisz, gdzie obecnie (2024) jest trenowana **największa** sieć neuronowa na świecie (+ jedno zdanie o tej sieci - zastosowanie)
1. Przeprowadź 7 symulacji z Playground Tensorflow (http://playground.tensorflow.org)



Zaobserwuj wyniki i działanie tej sieci neuronowej. Opisz (krótko!) jak zbudowana jest typowa sieć neuronowa.

2. Umieść w odpowiedzi wybrane 1-3 zrzuty ekranu z symulacji (z ustawionymi przez siebie parametrami)
3. Odpowiedz czym są i do czego służą wagi w sieciach neuronowych

4. Z wykorzystaniem biblioteki NumPy spróbuj zdefiniować:

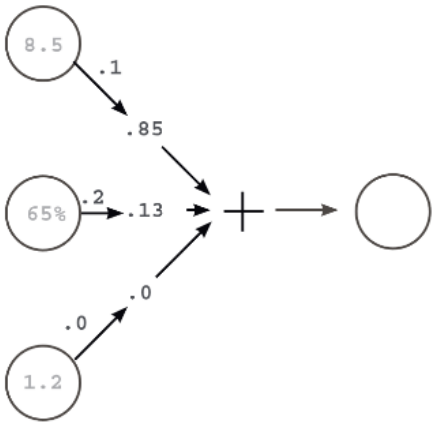
4.1 wektor [1, 2, 3, 4]
4.2 macierz 2 x 4 złożoną z samych zer
4.3 macierz 2 X 5 złożoną z liczb losowych między 0 i 1
4.4 wyjaśnij, co robi funkcja *dot*?

5. Implementacja **najprostszej sieci neuronowej** z Python służącej do prognozowania, czy pewna drużyna sportowa wygra rozgrywki (w dobie koronawirusa: e-Rozgrywki)
   5.1 Wprowadź zmienną (o nazwie: *weight*) i przypisz jej wartość 0.1
   5.2 Zdefiniuj prostą funkcję (o nazwie: *neural_network*) przyjmującą dwa parametry: *input* oraz utworzoną poprzednio wagę
   5.3 Funkcja powinna zwrócić zmienną (o nazwie: *pred*), w której przechowana będzie wartość mnożenia (*weight*input)*
   5.4 Prosta metoda powinna zostać przetestowana na następujących danych:
   `number_of_toes = [8.5, 9.5, 10, 9]` - średnia liczba rzutów dla drużyny przed rozpoczęciem e-Rozgrywek
   ```
   input = number_of_toes[0]
   pred = neural_network(input,weight)
   ```
   5.5 Wyświetl uzyskany wynik i wpisz go w odpowiedziach
   5.6 **Powtórz eksperyment** dla pozostałych danych 9.5, 10 oraz 9

6. Wykonamy teraz prognozę dla **wielu danych wejściowych** – w takim przypadku prosta sieć neuronowa powinna wykonywać bardziej dokładne prognozy
   6.1 Wstawimy następujące zbiory:
   ```
   toes =  [8.5, 9.5, 9.9, 9.0]
   wlrec = [0.65, 0.8, 0.8, 0.9]
   nfans = [1.2, 1.3, 0.5, 1.0]
   ```
   zbiór *toes* – bieżąca średnia udanych rzutów na gracza
   zbiór *wlrec* – bieżąca liczba zwycięstw (procentowo)
   zbiór nfans – liczba kibiców (w milionach)
   6.2 Zmodyfikujemy także naszą funkcję następująco:

```
def neural_network(input, weights):
        pred = w_sum(input,weights)
        return pred
```

   6.3 Dodajemy odpowiednie wagi (dla każdej danej wejściowej - połączenia)
   `weights = [0.1, 0.2, 0]`



6.4 Powyższy schemat prostej sieci neuronowej oraz zaznaczona w kodzie (na czerwono – poprzednia strona) nowa funkcja dokonująca konkretne obliczenia – **suma ważona danych wejściowych**:

Dane wejściowe	*wagi	=	Lokalne prognozy	
(8.5	* 0.1)	=	**0.85**	= prognoza *toes*
(0.65	* 0.2)	=	**0.13**	= prognoza *wlrec*
(1.20	* 0.0)	=	**0.0**	= prognoza *nfans*

Następnie: prognoza *toes* + prognoza *wlrec* + prognoza *nfans* = **prognoza finalna**

0.85 + 0.13 + 0.0 = 0.98

6.5 Spróbuj zaimplementować funkcję w_sum() przyjmującą dwa argumenty: a, b (wektory).
Wywołanie funkcji w kodzie: `pred = w_sum(input,weights)`
Zmienna `pred` przechowuje wartość wykonanej przez sieć neuronowej prognozy
Każda dana wejściowa jest mnożona przez wagę.
Ponieważ mamy wiele danych wejściowych, sumujemy odpowiadające im prognozy.

Mnożymy każdą danę wejściową przez odpowiadającą jej wagę i sumujemy wyniki (lokalne prognozy) – nazywamy to sumą ważoną danych wejściowych/sumą ważoną/iloczynem skalarnym.

6.6 Dodamy *input* do kodu:

```
input = [toes[0],wlrec[0],nfans[0]]
```

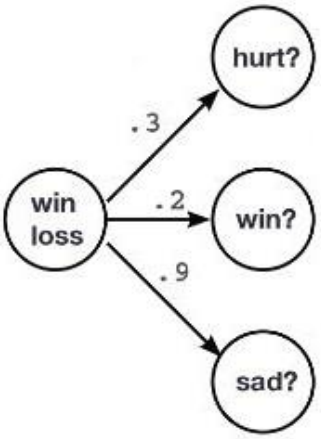
Input odpowiada wszystkim wpisom dla pierwszej rozgrywki w sezonie;

6.7 Na koniec wyświetl prognozę: `print(pred)`
Interfejs sieci neuronowej jest prosty. Akceptuje ona zmienną input jako infomację oraz zmienną weight jako wiedzę, po czym zwraca prognozę.

6.8 Spróbuj przenieść napisany kod w Pythonie (bez wykorzystania dodatkowych bibliotek) do kodu z wykorzystaniem jedynie biblioteki do obliczeń matematycznych **NumPy**

6.9 Następnie możemy spróbować zaimplementować **sieć neuronową wykonującą prognozy** z **wieloma danymi WYJŚCIOWYMI** – zamiast przewidywać czy pewna drużyna wygra eRozgrywki, przewidujemy czy nasi gracze są szczęśliwi (*win*), smutni (*sad*) oraz procentowy udział zawodników, którzy doznali kontuzji (*hurt*).

Prognozy wykonujemy **na bieżących stanach**: wygrana/przegrana (*win/loss*). Zauważamy, że w odróżnieniu od sieci neuronowej z wieloma wejściami i jednym wyjściem, w których prognoza jest niezaprzeczalnie powiązana, ta nowa sieć zachowuje się jak niezależne (trzy) komponenty, z których każdy odbiera te same dane wejściowe.



7. Ustawmy wagi: `weights = [0.3, 0.2, 0.9]`
   7.1 Kod funkcji:

```
def neural_network(input, weights):
    pred = ele_mul(input,weights)
    return pred
```

oraz dane (+ predykcja)

```
wlrec = [0.65, 0.8, 0.8, 0.9]

input = wlrec[0]

pred = neural_network(input,weights)
```

Prognozy są przechowywane jako **wektory liczb**

```
print(pred)
```

7.2 Wykonywanie mnożenia element po elemencie (funkcja o nazwie: *ele_mul*()):

Dane wejściowe	*	Wagi	=	Finalne prognozy	
(0.65	*	0.3)	=	**0.195**	= prognoza *hurt*
(0.65	*	0.2)	=	**0.13**	= prognoza *win*
(0.65	*	0.9)	=	**0.585**	= prognoza *sad*

Spróbuj zdefiniować funkcję wykonującą mnożenie element po elemencie:

```
def ele_mul(number,vector):
```

8. Na koniec – dokonajmy prognozy **z wieloma danymi wejściowymi (we) i wieloma danymi wyjściowymi (wy)**. Jak poprzednio, waga łączy każdy węzeł wejściowy z każdym węzłem wyjściowym, samo zaś prognozowanie odbywa się w zwykły sposób.

Obliczane są trzy niezależne sumy ważone dla danych wejściowych w celu uzyskania trzech prognoz.

Architekturę sieci można postrzegać z dwóch perspektyw: *możemy myśleć o niej jako o trzech wagach wychodzących z każdego węzła wejściowego albo o trzech wagach wchodzących do każdego węzła wyjściowego*.

8.1 Pusta sieć z wieloma wejściami i wyjściami:

```
weights = [ [0.1, 0.1, -0.3], #hurt?
            [0.1, 0.2, 0.0], #win?
            [0.0, 1.3, 0.1] ] #sad?

def neural_network(input, weights):
    pred = vect_mat_mul(input,weights)
    return pred
```

8.2 Wstawianie jednej obserwacji wejściowej:

```
toes =  [8.5, 9.5, 9.9, 9.0] #bieżąca śr. liczba rzutów
wlrec = [0.65,0.8, 0.8, 0.9] #procentowa liczba zwycięstw
nfans = [1.2, 1.3, 0.5, 1.0] #liczba kibiców (w mln)
```

Ten zbiór danych to bieżący stan na początku każdego meczu dla pierwszych czterech e-Rozgrywek w danym sezonie.

```
input = [toes[0],wlrec[0],nfans[0]]
```

Input odpowiada każdemu wpisowi dla pierwszego meczu w sezonie

```
pred = neural_network(input,weights)
```

8.3 Dla każdego wyjścia wykonujemy sumę ważoną danych wejściowych:

```
def w_sum(a,b):
    assert(len(a) == len(b))
    output = 0
    for i in range(len(a)):
        output += (a[i] * b[i])
    return output

def vect_mat_mul(vect,matrix):
    assert(len(vect) == len(matrix))
    output = [0,0,0]
    for i in range(len(vect)):
        output[i] = w_sum(vect,matrix[i])
    return output

def neural_network(input, weights):
    pred = vect_mat_mul(input,weights)
    return pred
```

Obliczenia:

#toes	% win	# fans	
(8.5*0.1)	+ (0.65*0.1)	+ (1.2*-0.3) = 0.555	= prognoza *hurt*
(8.5*0.1)	+ (0.65*0.2)	+ (1.2*0.0) = 0.98	= prognoza *win*
(8.5*0.0)	+ (0.65*1.3)	+ (1.2*0.1) = 0.965	= prognoza *sad*

Funkcja o nazwie vect_mat_mul() wykonuje *iteracje* przez wszystkie wiersze wag (każdy wiersz jest wektorem!) i wykonuje prognozę przy użyciu funkcji w_sum(). Zasadniczo wykonuje ona trzy kolejne sumy ważone i przechowuje ich prognozy w wektorze o nazwie *output*.

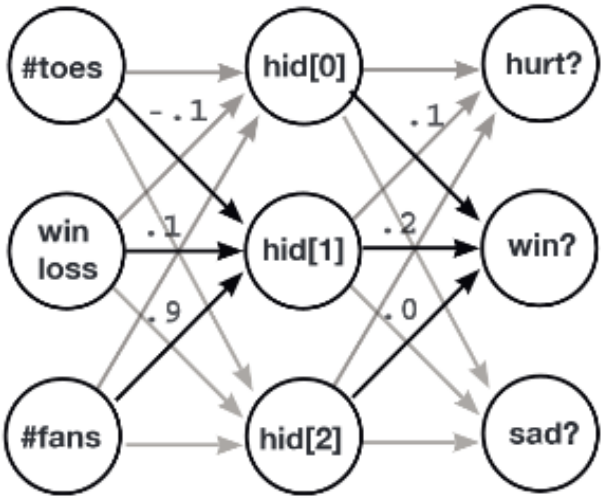
Zmienna *weight* jest listą wektorów – czyli macierzą. Powszechnie używane funkcje wykorzystują macierze. Jedną z tych funkcji jest mnożenie macierzy przez wektor. Ciąg sum ważonych jest właśnie tym: bierzemy wektor i wykonujemy iloczyn skalarny z każdym wierszem macierzy (*Osoby mające doświadczenie w algebrze liniowej zapewne znają bardziej formalną definicję, w której wagi są przechowywane jako wektory kolumnowe, a nie wierszowe, co zostanie skorygowane*)

Na koniec: `print(pred)`

9. Prognozowanie na podstawie prognoz… **sieci neuronowe ułożone jedna na drugą**
Można przyjąć wyjście z jednej sieci i przyjąć je jako wejście do innej sieci neuronowej. Daje to w efekcie dwa kolejne mnożenia wektorów i macierzy. Być może nie jest to jeszcze jasne dlaczego postępować w ten sposób – jednak niektóre zbiory danych (np. wykorzystywane w klasyfikowaniu obrazów) zawierają wzorce, które są zbyt złożone dla macierzy o pojedynczych wagach.

9.1 Pusta sieć neuronowa **z wieloma wejściami i wyjściami**

```
#toes   %win    #fans
ih_wgt = [ [0.1, 0.2, -0.1], #hid[0]
           [-0.1,0.1, 0.9], #hid[1]
           [0.1, 0.4, 0.1] ] #hid[2]

            #hid[0] hid[1] hid[2]
hp_wgt = [ [0.3, 1.1, -0.3], #hurt?
           [0.1, 0.2, 0.0], #win?
           [0.0, 1.3, 0.1] ] #sad?

weights = [ih_wgt, hp_wgt]
```



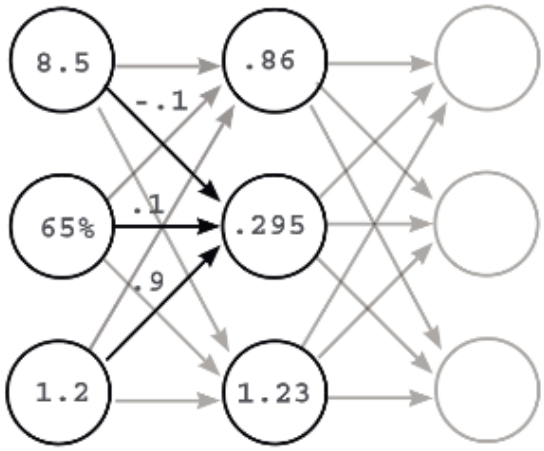
```
def neural_network(input, weights):

    hid = vect_mat_mul(input,weights[0])
    pred = vect_mat_mul(hid,weights[1])
    return pred
```

9.2 Prognozowanie warstwy ukrytej

```
toes =  [8.5, 9.5, 9.9, 9.0]
wlrec = [0.65,0.8, 0.8, 0.9]
nfans = [1.2, 1.3, 0.5, 1.0]
```

Dane wejściowe odpowiadają każdemu wpisowi dla pierwszego meczu w sezonie:

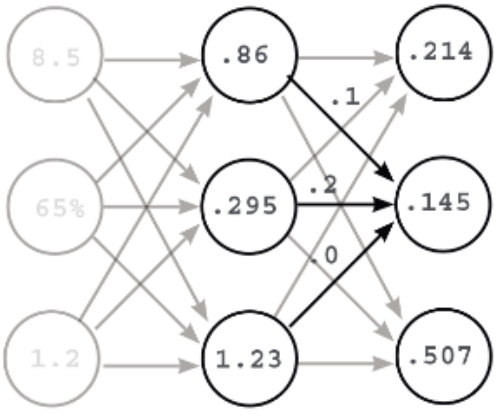
```
input = [toes[0],wlrec[0],nfans[0]]
```




```
pred = neural_network(input,weights)
```

```
def neural_network(input, weights):

    hid = vect_mat_mul(input,weights[0])
    pred = vect_mat_mul(hid,weights[1])
    return pred
```

9.3 Prognozowanie warstwy wyjściowej (i zachowanie prognozy):




```
def neural_network(input, weights):

    hid = vect_mat_mul(input,weights[0])
    pred = vect_mat_mul(hid,weights[1])
    return pred
```

```
toes =  [8.5, 9.5, 9.9, 9.0]
wlrec = [0.65,0.8, 0.8, 0.9]
nfans = [1.2, 1.3, 0.5, 1.0]
```

Dane wejściowe odpowiadają każdemu wpisowi dla

pierwszego meczu w sezonie: `input = [toes[0],wlrec[0],nfans[0]]`

```
pred = neural_network(input,weights)

print(pred)
```

9.4 Spróbuj napisać ten sam kod (powyżej) z wykorzystaniem biblioteki NumPy

10. To, co robiliśmy do tej pory nosi nazwę **propagacji w przód** (ang. *forward propagation*), w której sieć neuronowa przyjmuje dane wejściowe i wykonuje prognozę. Nazwa wynika stąd, że propagujemy aktywacje do przodu przez sieć, czyli od wejść sieci do jej wyjść. Aktywacje w przykładach są tymi wszystkimi liczbami, które nie są wagami i są unikatowe dla każdej prognozy. Inteligencja sieci jest uzależniona od wartości wag, którą jej przekażemy.