Projekt PSZT - sieci neuronowe

1.Definicja

Sieć neuronowa to struktura matematyczna składająca się z neuronów i połączeń między nimi. Działanie sieci polega na przyporządkowaniu danych wejściowych do danej klasy wyjściowej.

Proces przetwarzania danych można podzielić na 3 etapy:

1. Initialization
    - Create()
2. Forward
    - ComputeResult()
    - ComputeLoss()
3. Backward
    - ComputeGradient()
    - Learn()

Można też wczytać sieć z pliku (metodą readAll()) oraz zapisać do pliku (metodą writeAll()).

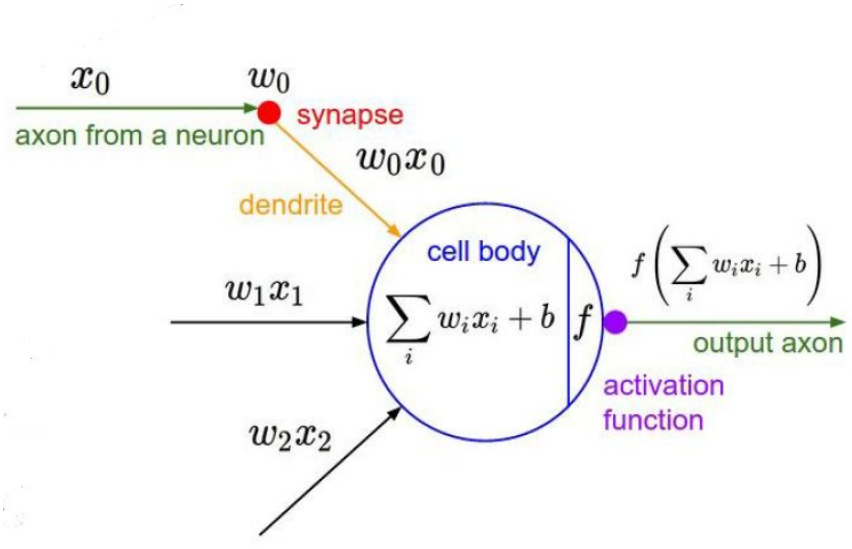
2.Wytłumaczenie etapów

Pierwszym etapem jest zainicjowanie wartości wag i biasów. Zaimplementowane są 3 metody(metoda Create()):

1. Random - losowe wartości
2. sqrt(2/n) - odchylenie standardowe = sqrt(2/n_{we})
3. Xavier - odchylenie standardowe = 4*sqrt(6/(n_{we}+n_{wy}))

gdzie n_{we} - ilość neuronów wejściowych, n_{wy} - ilość neuronów wyjściowych
Bias jest inicjalizowany z wartością 0.

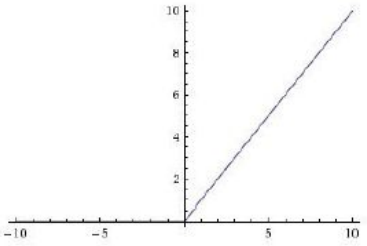
Podczas etapu Forward wartości na poszczególnych neuronach są obliczane zgodnie z tym wzorem(metoda
ComputeResult()):



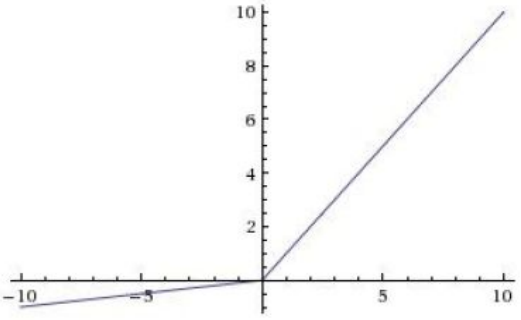
xi - wartość na neuronie
wi - waga na połączeniu
b - bias na neuronie

Zaimplementowane funkcje aktywacji to:

ReLU max(0,x)

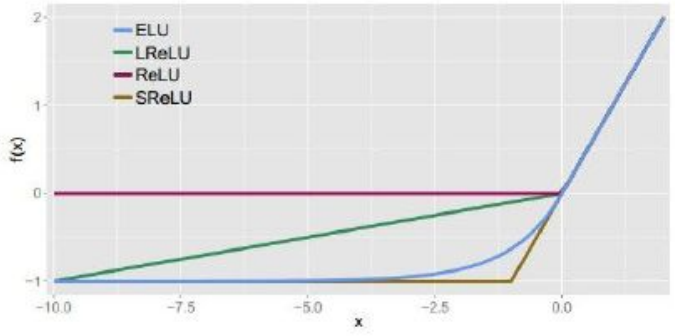


Leaky ReLU
max(0.1x, x)

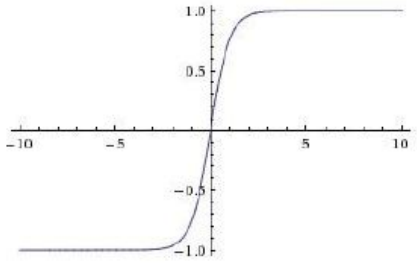


ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

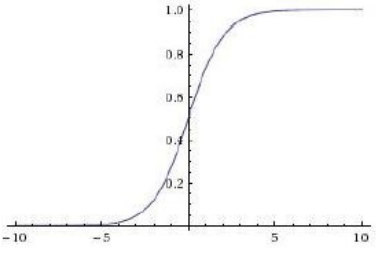


tanh tanh(x)



Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



Po przejściu całej sieci klasą przyporządkowania jest ta, na której jest największa wartość. Następnie obliczana jest regularyzacja L2 oraz strata jedną z 2 metod(metoda ComputeLoss()):

SVM $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Softmax $L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$

W kolejnym etapie jest obliczany gradient na każdym neuronie metodą Backpropagation (metoda ComputeGradient()) i następnie zmieniane są wagi i biasy na neuronach w zależności od gradientu.
Zaimplementowane są 2 metody uaktualniania(metoda Learn()):

Vanilla

```
x += - learning_rate * dx
```

Adam

```
# t is your iteration counter going from 1 to infinity
m = beta1*m + (1-beta1)*dx
mt = m / (1-beta1**t)
v = beta2*v + (1-beta2)*(dx**2)
vt = v / (1-beta2**t)
x += - learning_rate * mt / (np.sqrt(vt) + eps)
```

3a.Testy

Na konsultacjach pokazaliśmy Panu dopasowanie 81%, ale okazało się, że sieć wskazuje za każdym razem tą samą klasę, (0 - 1 klasa, 1 - 2 klasa, (-1) - taki sam wynik dla obu)

```
VERIFYING.. lines: 40527

0# 81.40745676% poprawnych (avg loss = 0.3784229366) 0: 32992/32992 (100%) 1: 0/7535 (0%) -1: 0/7535 (0%)
```

więc postanowiliśmy nauczyć sieć taka samą ilością danych z 1 klasy co z 2. - Teraz ilość danych do nauki wynosi 2 * 14019 = 28038, a testowych 40527.

```
LEARNING.. lines: 70000 (can be less because of evenly flag)

0# 56.40714719% poprawnych (avg loss = 0.8785375141) 0: 10032/14019 (71.56002568%) 1: 5784/14019 (41.25829232%) -1: 0/12223 (0%)
1# 56.47134349% poprawnych (avg loss = 0.8797152968) 0: 10094/14019 (72.00228262%) 1: 5740/14019 (40.94443256%) -1: 0/12205 (0%)
2# 56.49630871% poprawnych (avg loss = 0.8763896755) 0: 10135/14019 (72.29474285%) 1: 5706/14019 (40.70190456%) -1: 0/12198 (0%)
3# 56.47134349% poprawnych (avg loss = 0.8753224407) 0: 10145/14019 (72.36607461%) 1: 5689/14019 (40.58064056%) -1: 0/12205 (0%)
4# 56.45707764% poprawnych (avg loss = 0.8748256379) 0: 10149/14019 (72.39460732%) 1: 5681/14019 (40.52357515%) -1: 0/12209 (0%)
5# 56.42141303% poprawnych (avg loss = 0.8744792026) 0: 10152/14019 (72.41600685%) 1: 5668/14019 (40.43084385%) -1: 0/12219 (0%)
6# 56.4606441% poprawnych (avg loss = 0.8748528519) 0: 10133/14019 (72.2804765%) 1: 5698/14019 (40.64483915%) -1: 0/12208 (0%)
7# 56.47134349% poprawnych (avg loss = 0.8759440301) 0: 10111/14019 (72.12354662%) 1: 5723/14019 (40.82316856%) -1: 0/12205 (0%)
8# 56.44637826% poprawnych (avg loss = 0.880302114) 0: 10006/14019 (71.37456309%) 1: 5821/14019 (41.52221984%) -1: 0/12212 (0%)
9# 56.43211241% poprawnych (avg loss = 0.8750268997) 0: 9877/14019 (70.45438334%) 1: 5946/14019 (42.41386689%) -1: 0/12216 (0%)
10# 56.43567888% poprawnych (avg loss = 0.8744838229) 0: 9730/14019 (69.40580641%) 1: 6094/14019 (43.469577%) -1: 0/12215 (0%)
loss < 0.8785375141, 0.8744838229 > : fit < 56.40714719, 56.49630871 >%

VERIFYING.. lines: 40527

0# 61.56636317% poprawnych (avg loss = 0.7596219653) 0: 20829/32992 (63.13348691%) 1: 4122/7535 (54.70471135%) -1: 0/15576 (0%)
```

Wyniki okazały się dużo gorsze, więc przetestowaliśmy takie same warunki w sieci zaimplementowanej za pomocą biblioteki tflearn.

```
Training Step: 46197  | total loss: 0.68950 | time: 1.197s
| Adam | epoch: 100 | loss: 0.68950 - acc: 0.5397 -- iter: 29376/29568
Training Step: 46198  | total loss: 0.68664 | time: 1.199s
| Adam | epoch: 100 | loss: 0.68664 - acc: 0.5545 -- iter: 29440/29568
Training Step: 46199  | total loss: 0.68606 | time: 1.201s
| Adam | epoch: 100 | loss: 0.68606 - acc: 0.5522 -- iter: 29504/29568
Training Step: 46200  | total loss: 0.68533 | time: 2.493s
| Adam | epoch: 100 | loss: 0.68533 - acc: 0.5532 | val_loss: 0.68266 - val_acc: 0.5903 -- iter: 29568/29568
--
```

Dopasowanie 59% po 100 epokach na 30000 danych mówią samo za siebie - dane są zbyt losowe. - lub pomijamy zbyt wiele kolumn z excela?

Na stronie z danymi w zakładce dyskusja znajduje się wiele wykresów przedstawiających wpływ każdej danej na ostateczny wynik.

PART III: PREDICTING WHETHER A PERSON WILL BE SHOWING UP

In this section I am going to try and predict the Show-Up/No-Show status based on the features which show the most variation in probability of showing up. They are:

1. Age
2. Diabetes
3. Alchoholism
4. Hypertension
5. Smokes
6. Scholarship
7. Tuberculosis

Wszystkie z tych danych pobieramy z excela, więc wzięcie pod uwagę innych danych wprowadziłoby znikomą poprawę.
Na tej samej stronie zostało przedstawione dopasowanie na poziomie

```
Accuracy: 71.0 %
```

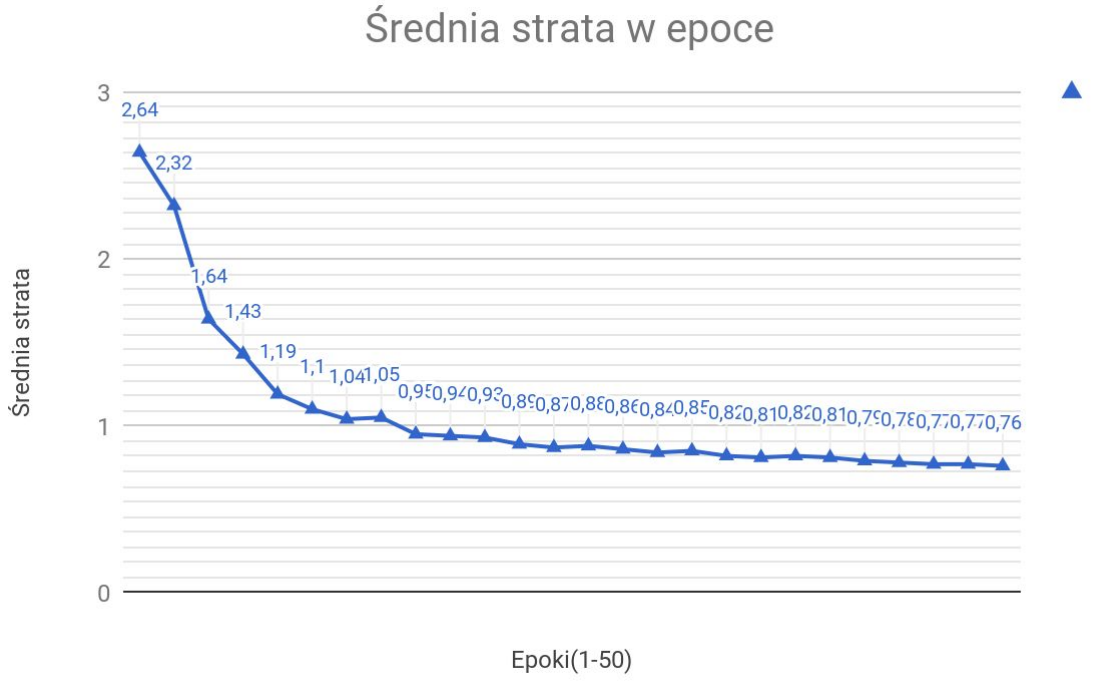
Lecz w komentarzach zwrócili mu uwagę, że faworyzuje jedną z klas

Have you checked whether your model actually predicts? The percentage of patients showing up is around 70%. So a simple model that always returns "Show-up" would automatically have an accuracy of 70%.

Temat nie został zbytnio rozbudowany, więc raczej nikomu nie udało się uzyskac lepszych wyników, więc chyba można założyć, że to problem z danymi.

3b. Testy

Postanowiliśmy przetestować nasz model na innych danych i wybraliśmy przykładowe dane z biblioteki pythona “titanic_dataset.csv”.



Po 1000 epokach udało się osiągnąć taki wynik:

```
987# 83.89380531% poprawnych (avg loss = 0.4386700525) 0: 254/282 (90.07092199%) 1: 220/282 (78.0141844%) -1: 0/91 (0%)
988# 84.24778761% poprawnych (avg loss = 0.4381366901) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
989# 84.24778761% poprawnych (avg loss = 0.4378990406) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
990# 84.24778761% poprawnych (avg loss = 0.438766768) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
991# 84.24778761% poprawnych (avg loss = 0.4381763461) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
992# 84.24778761% poprawnych (avg loss = 0.4372880964) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
993# 84.24778761% poprawnych (avg loss = 0.4379715932) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
994# 84.24778761% poprawnych (avg loss = 0.437611817) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
995# 84.24778761% poprawnych (avg loss = 0.4367963503) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
996# 84.07079646% poprawnych (avg loss = 0.4485570331) 0: 253/282 (89.71631206%) 1: 222/282 (78.72340426%) -1: 0/90 (0%)
997# 83.71681416% poprawnych (avg loss = 0.4527851841) 0: 254/282 (90.07092199%) 1: 219/282 (77.65957447%) -1: 0/92 (0%)
998# 84.07079646% poprawnych (avg loss = 0.4402401868) 0: 254/282 (90.07092199%) 1: 221/282 (78.36879433%) -1: 0/90 (0%)
999# 84.24778761% poprawnych (avg loss = 0.4395944071) 0: 255/282 (90.42553191%) 1: 221/282 (78.36879433%) -1: 0/89 (0%)
1000# 83.89380531% poprawnych (avg loss = 0.4386227671) 0: 255/282 (90.42553191%) 1: 219/282 (77.65957447%) -1: 0/91 (0%)
loss < 27.28367632, 0.4386227671 > : fit < 50.44247788, 84.60176991 >%

VERIFYING.. lines: 409

0# 76.52811736% poprawnych (avg loss = 5.409941863) 0: 203/263 (77.18631179%) 1: 110/146 (75.34246575%) -1: 0/96 (0%)
```

Dopasowanie na poziomie 76% wygląda dobrze, metoda dropout mogłaby dać jeszcze lepsze wyniki, ale niestety nie jest zaimplementowana.

4.Obsługa

Po włączeniu programu odpali się przykładowa nauka i gdy się zakończy mamy do wyboru 8 komend

1. `NEW SCHUFF ACTIV INIT UPDATE LOSS` - ustawienie flag oraz rodzaje funkcji następnej nauki. 0-1(czy tworzyć nową sieć) 0-3(mieszanie danych) 0-5(funkcje aktywacji) 0-2(funkcje inicjalizacji) 0-1(metoda aktualizacji) 0-1(funkcja straty)
2. `EDIT(-)`
   - `test_lines iterations verify_lines ratio lambda decay evenly` - Parametry nauki. decay(co ile epok ma zmniejszać się ratio i lambda 2-krotnie), evenly(czy klasy mają zostać zbalansowane)
3. `BACKUP(b)` - wczytanie sieci sprzed nauki
4. `SAVE(s/[0-9]s)` - zapisanie do pliku [0-9] lub do folderu głównego
5. `LOAD(l/[0-9]l)` - wczytanie z pliku [0-9] lub z folderu głównego
6. `FULL(f/lf)` - f - sprawdzenie na wszystkich danych testowych (lf - nauka + sprawdzenie)
7. `RESET(r)` - zeruje parametry potrzebne do Adam'a
8. `DIVIDE((r,l, )/)` - dzieli ratio/lambde/oba przez 2