Cezary Storczyk, 263486	Data: 24 V 2024
	Prowadzący: dr inż. Piotr Ciskowski
Projekt: Sieci Neuronowe, Temat: sieć dwuwarstwowa problem XOR	

1. Przedstawnienie modelu

Aby rozwiązać problem XOR zbudowałem sieć neuronową która uczy się rozwiązywać ten problem. Sieć ma dwa wejścia oznaczające dwie cechy x1 i x2 . Cechy mogą mieć kombinacje [0,0, 0,1, 1,0, 1,1]. Dokładna inicjalizacja wygląda następująco:

```
P = np.array([[0,0,1,1],
              [0,1,0,1]] )  # Wejścia sieci

T = np.array([0,1,1,0])   # żądane wyjście sieci
```

Rysunek 1: Wejścia problemu XOR oraz oczekiwane wyjścia

Inicjalizacja sieci polega na stworzeniu macierzy wag o odpowiednich wymiarach i wartościach początkowych. Wartości początkowe, dla których sieć uczy się dobrze, to wartości losowe z przedziału -0.1 do 0.1. Jest możliwość użycia inicjalizacji Xaviera, natomiast dla tak małej sieci jest to wręcz niestosowne.

```
W1_matrix, W2_matrix = init2(2, 2, 1)

print(W1_matrix)
print(W2_matrix)
print(P[:,0])
```

Executed at 2024.05.20 12:38:34 in 6ms

```
[[ 0.01253961 -0.07473697 -0.03794824]
 [-0.07142703  0.07194731  0.07364697]]
[[-0.02861943  0.01743309  0.00195038]]
[0 0]
```

Rysunek 2: Najprostsza sieć która jest zdolna nauczyć się problemu XOR

2. Propagacja w przód

Do wejść, i warstwy ukrytej dodajemy bias równy -1. Wynik dot product z macierzy wag pierwszej warstwy sieci i wartości wejść do sieci to wektor o ilości neuronów w pierwszej warstwie: U1. Korzystam z sigmoidalnej funkcji aktywacji: Y1. Drugi dot product to iloczyn wartości macierzy wag drugiej warstwy

```
beta = 5
X1 = np.append(X, -1)
U1 = np.dot(W1, X1)
Y1 = 1 / (1 + np.exp(-U1 * beta))
```

Rysunek 3: Propagacja w przód

sieci z wartościami Y1. Wynikiem jest skalar, przewidujący wartość wyniku XOR. Jest to liczba która wraz z kolejnymi iteracjami uczenia sieci będzie się poprawiać w stronę coraz dokładniejszych przybliżeń rzeczywistej wartości XOR.

3. Propagacja wstecz

Jest to nasza funkcja ucz2.py. Aby móc aktualizować wagi sieci, potrzebujemy mieć algorytm który umie patrzeć jak zmiana wag w poprzedzających częściach sieci wpływa zmianę wyniku. Do tego służy nam pochodna. Liczymy więc ”derivative with respect to : wagi w pierwszej warstwie” oraz ”derivative with respect to: wagi w drugiej warstwie”.

```
cost = True_value[random_sample] - Y2
derivative_2warstwa = cost * beta * Y2 * (1-Y2)
dj_dW2 = wspUcz * np.outer(derivative_2warstwa, Y1)
```

Rysunek 4: Propagacja w przód

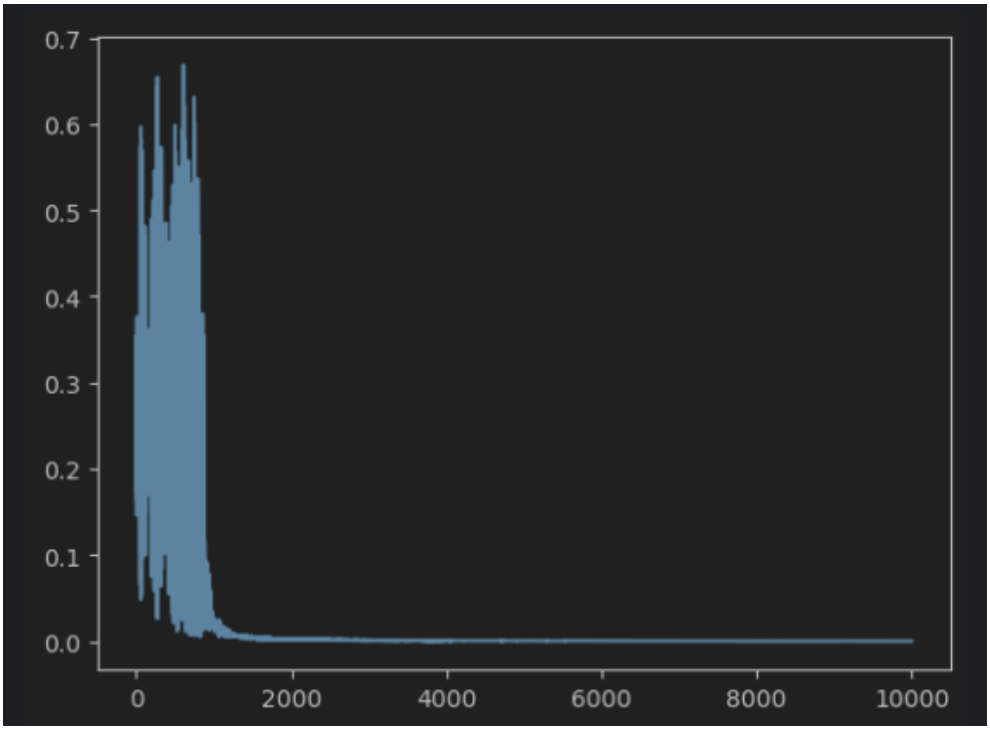
- Wyliczenie pochodnych pozwala nam na aktualizację wag. Im mniejsze pochodne tym model coraz bardziej ”umie” rozwiązywać problem. Aby proces uczenia był na dalszych etapach efektywny, stosuje się zmiejszanie learning rate w kolejnych krokach.
- Dodatkowo aby nie utknąć w minimach lokalnych stosuje się momentum, kóre działa jak siła bezwładności rozpędzonego samochodu (jeśli skończy się paliwo, i jesteśmy w dołku, to może uda nam się jeszcze tym rozpędzonym samochodem wydostać z dołka.
- Do wcześniejszego przerwania uczenia zastosowałem minimalny błąd mse który należy osiągnąć aby sieć nauczyła się problemu. Dzięki temu, nie musimy uczyć modelu przez wszystkie iteracje.
- MSE jest różnicą między wartościami rzeczywistymi a tymi które dał model. Zmniejszanie MSE oznacza polepszanie modelu.

```
W1po, W2po, mse_history = ucz2(W1_matrix, W2_matrix, P, T, 10000, 1e-9)
Executed at 2024.05.20 12:38:36 in 794ms
```

Rysunek 5: Rozpoczęcie procesu uczenia

4. Wykres błędów:

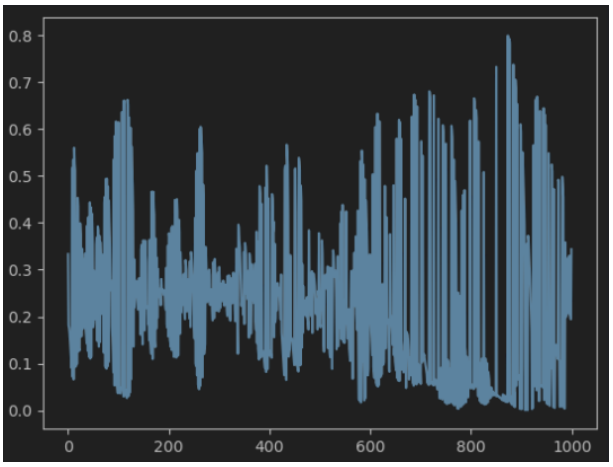
Wykres błędów pokazuje nam jak zmieniała się efektywność modelu w trakcie uczenia. W każdej iteracji zapisywałem wynik MSE, aby następnie zobaczyć efektywność modelu:



Rysunek 6: MSE przez kolejne 10000 iteracji nauki modelu

5. Wyniki modelu

Ilość iteracji: 1000. Widać że błąd MSE oscyluje wokół pewnych wartości, model na tym etapie nie daje jeszcze coraz lepszych wyników, chociaż wartości wag dają mu coraz większy potencjał do rozpoznawania.



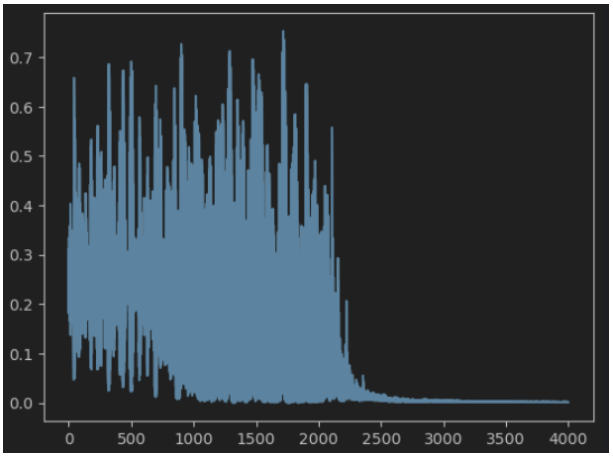
Rysunek 7: Wykres MSE

Model nie nauczył się problemu XOR. Wyniki liczbowe: Z wyników liczbowych możemy zauważyć że wejście (0,0) jest prawie bezbłędnie klasyfikowane 0 (0.05 jest bliskie 0).

```
dla wejścia XOR(0,0) = [0.05730302]   (oczekiwane 0)
dla wejścia XOR(0,1) = [0.55952665]   (oczekiwane 1)
dla wejścia XOR(1,0) = [0.55633054]   (oczekiwane 1)
dla wejścia XOR(1,1) = [0.58625896]   (oczekiwane 0)
```

Rysunek 8: Wartości liczbwe

Ilość iteracji: 4000. Widać że błąd MSE zmniejszył się mocno po 2500 tysiącu iteracji. Ten spadek oznacza że model przełamał barierę problemu i jest już w stanie rozdzielać poprawnie problem XOR (chociaż jeszcze nie ze 100% dokładnością.



Rysunek 9: Wykres MSE

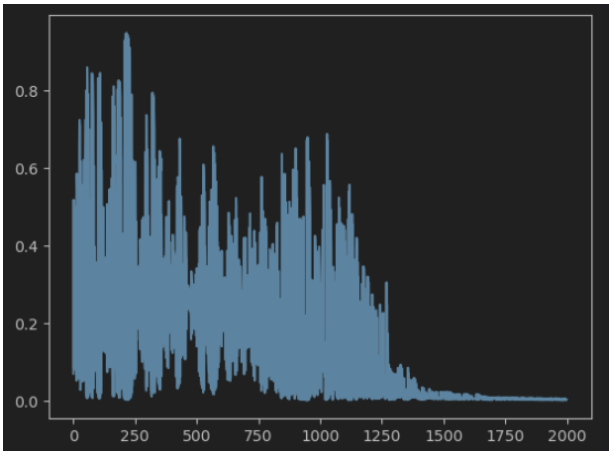
Model nauczył się XOR. Wyniki liczbowe: Teraz wszystkie wejścia są rozpoznawane z większą pewnością.

```
dla wejścia XOR(0,0) = [0.03157516]   (oczekiwane 0)
dla wejścia XOR(0,1) = [0.96734395]   (oczekiwane 1)
dla wejścia XOR(1,0) = [0.96752579]   (oczekiwane 1)
dla wejścia XOR(1,1) = [0.04262606]   (oczekiwane 0)
```

Rysunek 10: Wartości liczbwe

6. Więcej neuronów

Zwiększenie ilości neuronów pomaga w nauce sieci.



Rysunek 11: 16 neuronów w warstwie ukrytej

Dzięki zwiększeniu ilości neuronów sieć uczy się szybciej. Oznacza to że wyciąga więcej wniosków z każdej iteracji.

7. WNIOSKI

- Wraz ze zwiększaniem ilości iteracji, model poprawia swoje wyniki
- Istnieje granica w której model drastycznie poprawia swoje wyniki
- Dzięki większej ilości iteracji, model uczy się coraz lepiej problemu XOR
- Dla 10000 iteracji model uzyskuje wyniki : 0.0095, 0.9897, 0.9896, 0.0125
- zwiększanie ilości neuronów w warstiwe ukrytej zwiększa "umiejętności" sieci do uczenia się.
- zwiększenie ilości neuronów do bardzo dużej wartości jak 160 powoduje skakanie MSE, brak rezultatów z nauki