Laboratorium 5 - sprawozdanie

Wstęp do sztucznej inteligencji

Patryk Będkowski Marcin Grabysz

03.01.2022

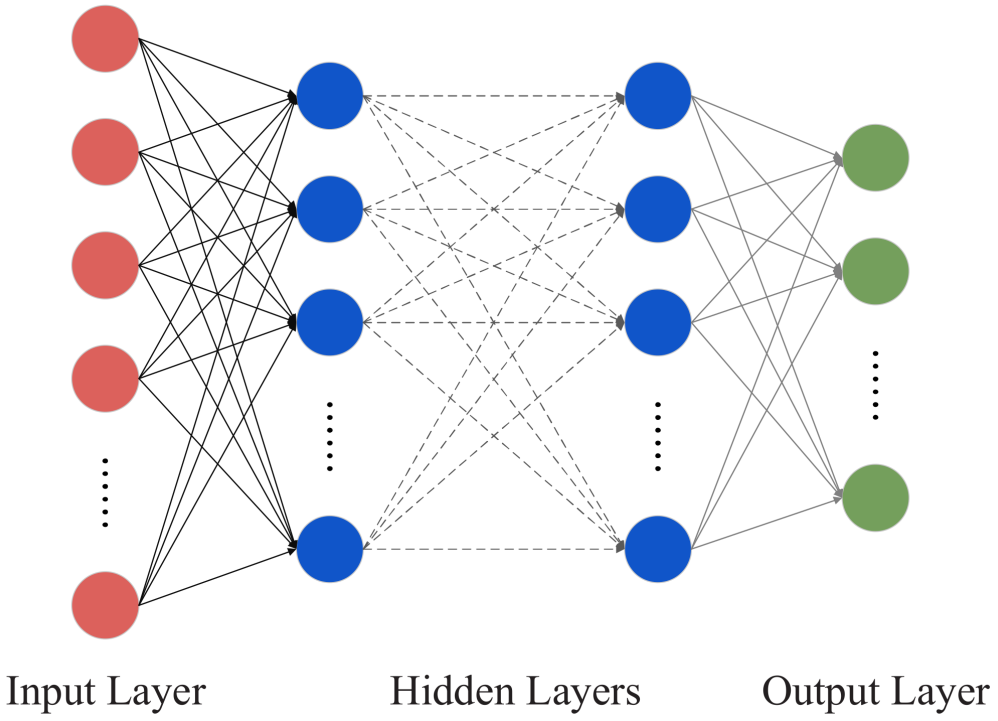
1 Treść zadania

W ramach piątych ćwiczeń będą Państwo musieli zaproponować architekturę, zaimplementować, wytrenować i przeprowadzić walidację sieci neuronowej do klasyfikacji ręcznie pisanych cyfr. Zbiór danych do użycia: MNIST.

2 Schemat sieci neuronowej

Poniższa grafika przedstawia przykładowy model sieci neuronowej. Określony przez nas najlepszy model składa się z czterech warstw:

- warstwa wejściowa: posiada 784 neurony (każdy jeden przyjmuje wartość jednego piksela)
- pierwsza warstwa ukryta: posiada 30 neuronów
- druga warstwa ukryta: posiada 15 neuronów
- warstwa wyjścia: warstwa aktywacji posiadająca 10 neuronów



Rysunek 1: Schemat sieci neuronowej

3 Implementacja

3.1 Wykorzystane narzędzia

Algorytmy i funkcje służace badaniu funkcji zaimplementowane są w języku Python w wersji 3.8.10. Do przygotowania danych wykorzystane zostały pewne funkcje z bibliotek `keras` i `sklearn`. Funkcje pracujące na macierzach zostały zaimplementowane przy użyciu modułu `numpy`, a do zapisywania wyników badania została użyta klasa `Dataframe` z biblioteki `pandas`.

3.2 Opis implementacji

3.2.1 Moduł file_management.py

W tym module zaimplementowane są podstawowe funkcje przygotowujące dane do użycia. Do nauczenia sieci potrzebny jest wektor wejściowy

(reprezentujący kolejne piksele badanej grafiki) oraz oczekiwany wektor wynikowy (poprawna odpowiedź dla danego przypadku). Uczenie się sieci przebiega dzięki porównywaniu wygenerowanego przez sieć wektora z wektorem oczekiwanym.

3.2.2 Moduł Layers.py

W module Layers.py zaimplementowana jest abstrakcyjna klasa `Layer` reprezentująca warstwę sieci neuronowej oraz klasy dziedziczące:

- `Dense` - reprezentuje warstwę ukrytą lub wejściową (nie różnią się w przyjętej implementacji).
- `Activation` - reprezentuje funkcję aktywacji. W teorii sieci neuronowych aktywacja nie jest przedstawiana jako osobna warstwa, ale na potrzeby lepszego rozłożenia problemu można ją tak potraktować (należy jednak pamiętać o tym, że każdej instancji klasy `Dense` musi towarzyszyć jej funkcja aktywacji).
- `Softmax` - warstwa wyjściowa sieci neuronowej.

3.2.3 Moduł main.py

W tym module zaimplementowana jest właściwa sieć neuronowa - klasa `Neural_net`. W module znajdują się także funkcje służące testowaniu sieci.

3.3 Analiza zbioru treningowego

Zbiór treningowy składa się z 60 000 próbek ręcznie napisanych cyfr. Poniżej przedstawiono liczbę wystąpień każdej cyfry:

0	5923
1	6742
2	5958
3	6131
4	5842
5	5421
6	5918
7	6265
8	5851
9	5949

Tabela 1: Liczba wystąpień cyfr w zbiorze treningowym

Widzimy, że w zbiorze treningowym znajduje się najmniej próbek dla cyfry 5 (ten fakt okaże się przydatny w dalszej analizie), natomiast najwięcej grafik posiada cyfra 1. Każda próbka została przedstawiona jako grafika 28 x 28 pikseli w odcieniach szarości, gdzie piksel posiada wartość liczbową z zakresu 0-255.

4 Badanie implementacji

Ze względu na czas uczenia się sieci neuronowej (całe badanie wykonywało się przez około 55 minut), postanowiliśmy przeprowadzić badanie implementacji według następującego schematu:

1. nauczenie i przetestowanie sześciu różnych sieci (trzech o jednej warstwie ukrytej i trzech o dwóch warstwach ukrytych) ze współczynnikiem uczenia równym 0,1 oraz liczbą epok równą 10

2. wybranie najlepszego modelu i przetestowanie różnych wartości współczynnika uczenia i liczby epok dla tego modelu

Testowanie zostało przeprowadzone na zbiorze testującym liczącym 10 000 próbek. Otrzymane wyniki przedstawione są w poniższych tabelach. Parametry w tabeli to kolejno: liczba porządkowa, liczba epok, współczynnik uczenia, liczba warstw ukrytych, liczba neuronów w pierwszej wartwie ukrytej, liczba neuronów w drugiej warstwie ukrytej, procent poprawnych klasyfikacji, czas uczenia i testowania (ze względu na szerokość tabeli, tytuły kolumn musiały zostać napisane skrótowo).

L.p.	Epochs	Learn. rate	Hid. lay.	Lay. 1	Lay. 2	Correct. (%)	Time (s)
0	10	0,1	1	16	-	81,47	138,13
1	10	0,1	1	25	-	82,51	163,70
2	10	0,1	1	40	-	83,88	195,76
3	10	0,1	2	30	15	92,46	211,89
4	10	0,1	2	15	15	86,06	187,50
5	10	0,1	2	15	30	91,44	191,64

Tabela 2: Różne kombinacje warstw i liczby neuronów

L.p.	Epochs	Learn. rate	Hid. lay.	Lay. 1	Lay. 2	Correct. (%)	Time (s)
0	10	0,05	2	30	15	90,59	214,42
1	10	0,10	2	30	15	92,12	218,46
2	10	0,15	2	30	15	92,77	216,26
3	10	0,20	2	30	15	93,64	213,83

Tabela 3: Różne wartości współczynnika uczenia

L.p.	Epochs	Learn. rate	Hid. lay.	Lay. 1	Lay. 2	Correct. (%)	Time (s)
0	5	0,2	2	30	15	91,56	109,85
1	10	0,2	2	30	15	93,44	217,01
2	15	0,2	2	30	15	93,91	328,93
3	20	0,2	2	30	15	94,08	436,93

Tabela 4: Różne liczby epok

5 Wnioski

5.1 Wnioski ogólne

Z tabeli 2 wynika, że najlepszy model ma odpowiednio 30 i 15 neuronów w kolejnych warstwach ukrytych. W dalszych testach posłużyliśmy się modelem o tych parametrach. Najlepszy osiągnięty wynik to skuteczność na poziomie 94%. Warto także zauważyć, że skuteczność najlepszego modelu nie

spadła ani razu poniżej 90%, zaś dla wszystkich badanych modeli - poniżej 80%.

Co ważne, w badanych przypadkach zawsze modele z dwoma warstwami ukrytymi okazywały się lepsze od tych z jedną warstwą, nawet wtedy gdy sumaryczna liczba neuronów sieci z dwoma warstwami była mniejsza (porównanie przypadków 2. i 4. z tabeli 1.). Generalnie dla sieci o takiej samej liczbie warstw, większa liczba neuronów przekładała się na większą skuteczność (choć nieliniowo).

Należy jednak zachować pewne ograniczone zaufanie do otrzymanych wyników. Choć algorytm uczenia się sam w sobie jest deterministyczny, niedeterministyczne jest losowe zainicjowanie wag sieci (a takie założenie przyjęliśmy w implementacji) oraz pomieszanie danych początkowych (choć to akurat było takie samo dla wszystkich testów). Żeby wyeliminować z badania "łut szczęścia", należałoby przeprowadzić każdy z testów pewną ilość razy i podać przynajmniej średnią arytmetyczną - zrezygnowaliśmy jednak z tego podejścia, mając na uwadze czas wykonywania się algorytmów.

5.2 Macierz błędów sieci neuronowej

Sprawdzono, które próbki ze zbioru testowego zostały źle sklasyfikowane przez wytrenowaną sieć. W otrzymanej macierzy błędów kolumny reprezentują faktyczne cyfry, a wiersze - predykcje.

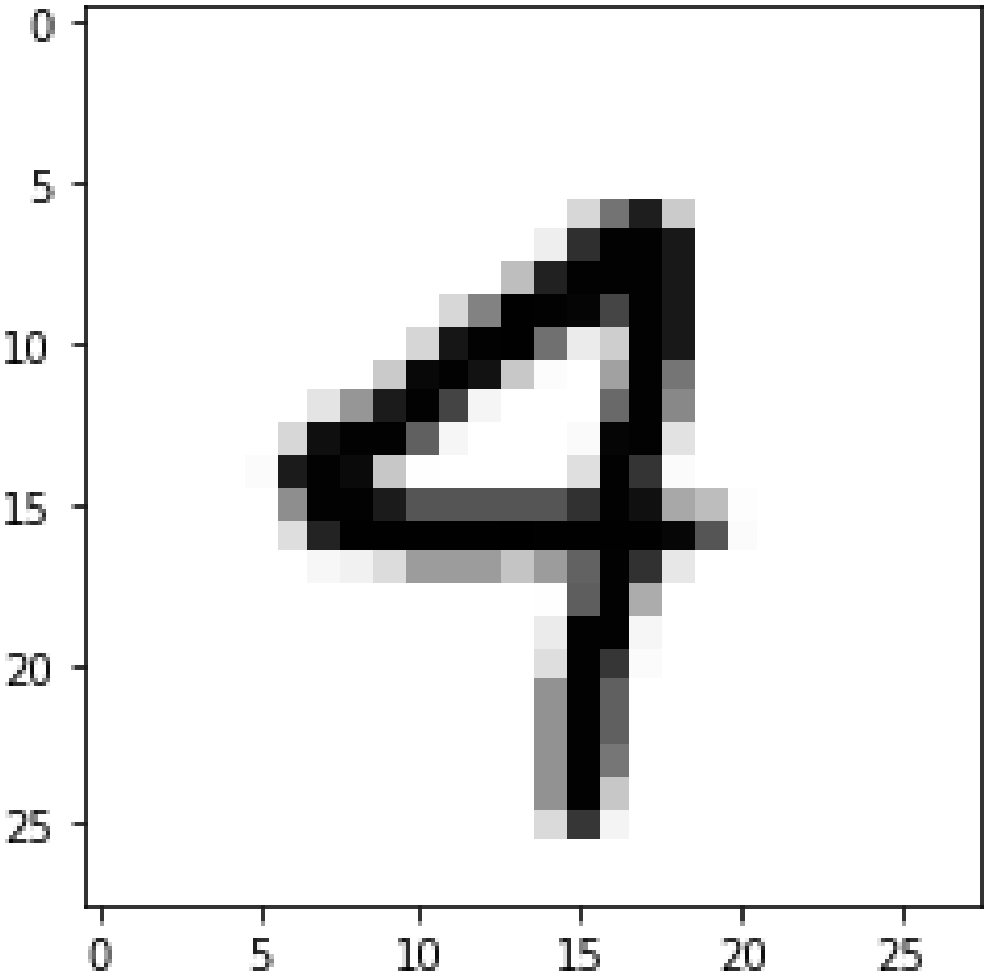
	0	1	2	3	4	5	6	7	8	9
0	959	2	18	8	0	12	12	0	12	10
1	0	1117	2	0	0	2	4	10	6	6
2	2	4	969	26	12	4	10	34	10	2
3	2	8	32	962	0	38	0	12	26	26
4	0	2	20	0	933	6	12	4	14	34
5	8	4	2	18	0	817	22	0	8	6
6	20	4	10	4	16	30	918	4	6	0
7	6	2	20	14	6	6	0	983	8	14
8	4	10	22	20	8	38	18	2	925	20
9	0	0	0	6	56	14	2	24	8	950

Tabela 5: Macierz błędów wytrenowanej sieci

Okazuje się, że najwięcej razy zostały pomylone cyfry 4 i 9. Takowych przypadków jest 56.

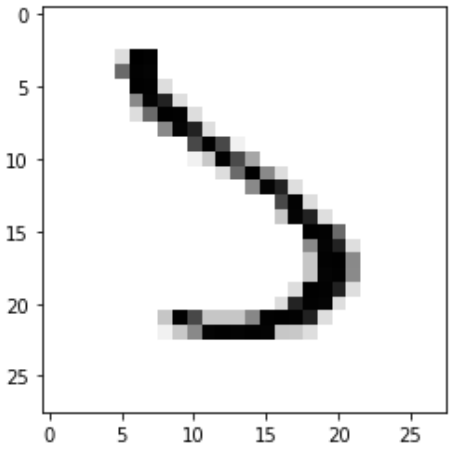
5.3 Przykłady zbioru MNIST sklasyfikowane niepoprawnie

Przykład grafiki przedstawiający cyfrę 4, która została sklasyfikowana przez sieć jako cyfra 9:

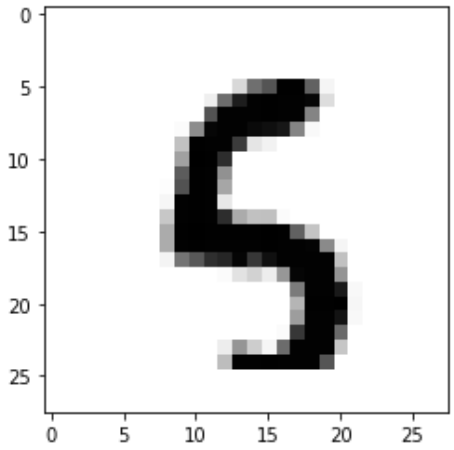


Rysunek 2: Cyfra 4 zaklasyfikowana jako 9

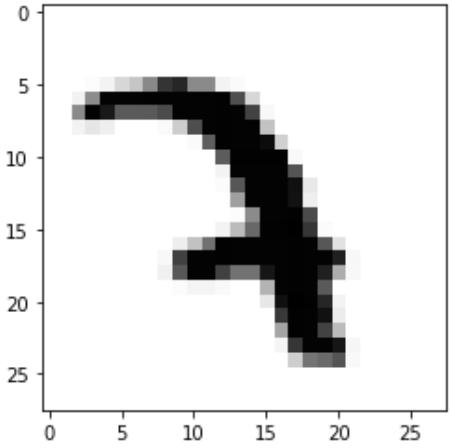
Poniżej przedstawiono kolejne grafiki błędnie sklasyfikowanych cyfr:



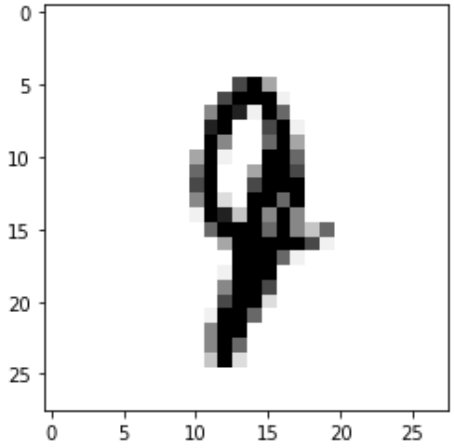
(a) 5 zakwalifikowane jako 3



(b) 5 zakwalifikowane jako 8



(c) 7 zakwalifikowane jako 2



(d) 9 zakwalifikowane jako 4

Rysunek 3: Przykłady błędnie zaklasyfikowanych cyfr

Większość z przedstawionych próbek przedstawia grafiki cyfr, które nie są napisane starannie. Nie dziwi więc, że wytrenowany model nie poradził sobie z ich klasyfikacją.

Poniższa tabela przedstawia ile razy dana cyfra pojawiła się w zbiorze testowym i ile razy została zaklasyfikowana niepoprawnie:

Cyfra	Liczba błędnych klasyfikacji	Liczba wystąpień w zb. testowym
0	42	1033
1	36	1147
2	126	1073
3	96	1106
4	98	1025
5	150	885
6	80	1012
7	90	1059
8	98	1067
9	118	1060

Tabela 6: Liczba błędnych klasyfikacji i liczba wystąpień dla każdej cyfry

Z powyższej analizy możemy wywnioskować, że nasz model najgorzej radzi sobie z klasyfikacją cyfry 5. Posiada ona największą liczbę błędnie sklasyfikowanych próbek. Dodatkowo na powyższej grafice przedstawiającej błędnie sklasyfikowaną cyfrę 5 (Rysunek 3b), widać że jest ona napisana wyraźnie i starannie. Co więcej, cyfra 5 jest zaklasyfikowana błędnie najczęściej co do wartości bezwzględnej nawet pomimo tego, że występuje najrzadziej, tj. sieć neuronowa najmniej razy w ogóle podejmowała próbę zaklasyfikowania tej cyfry.

Wyjaśnieniem tego zjawiska może być fakt, że zbiór testowy odzwierciedla zbiór treningowy (Tabela 1), czyli cyfra 5 występowała najrzadziej również podczas treningu. Oznacza to, że sieć miała najmniej okazji aby nauczyć się rozpoznawać tę cyfrę, co stanowi dla niej pewne usprawiedliwienie.

Należy tu zauważyć dokładną analogię do faktu, że cyfra 1 została błędnie sklasyfikowana najmniejszą ilość razy - występuje najczęściej zarówno w zbiorze treningowym, jak i testowym.

6 Porównanie z modelem Keras

Dodatkowo, postanowiliśmy sprawdzić jak *accuracy* naszej sieci ma się w stosunku do tej osiąganej przez model z biblioteki Keras. W tym celu zmodyfikowaliśmy kod z oficjalnej strony tensorflow tak, aby przetestować sieć o parametrach, które my uznaliśmy za najskuteczniejsze (30/15 neuronów w dwóch warstwach) i uruchomiliśmy za pomocą Google Collab (w/w strona udostępnia taką możliwość).

Wynik osiągnięty przez sieć z Kerasa po 10 epokach wynosi 95,73% i jest to tylko nieznacznie więcej, niż osiągnęła nasza sieć (dla przypomnienia - 94,08%).

7 Podział zadań

Generalnie uważamy, że obaj włożyliśmy w projekt podobną ilość pracy, a nad większością zagadnień zastanawialiśmy się wspólnie.

Patryk Będkowski:

- implementacja abstrakcyjnej klasy `Layer` i pochodnych
- preprocessing danych
- analiza zbioru treningowego
- przykłady błędnych klasyfikacji
- macierz błędów

Marcin Grabysz:

- implementacja main.py
- implementacja funkcji aktywacji
- testowanie modeli
- analiza wyników i porównanie z modelem z Kerasa
- redagowanie sprawozdania