

Sprawozdanie z projektu ze Sztucznej Inteligencji

Porównanie metod uczenia maszynowego w problemie MNIST i jego pochodnych

Filip Gołaś s188776
Damian Jankowski s188597
Mikołaj Storoniak s188806

May 21, 2023

1 Opis problemu

Tutaj będzie bardzo ładny opis problemu.

2 Opis porównanych modeli

2.1 Drzewa

2.1.1 Drzewo decyzyjne

2.1.2 Las losowy

2.2 Sieci neuronowe

Sieć neuronowa jest modelem statystycznym czerpiącym inspirację z natury i działania układów nerwowych żywych organizmów. Modele te są stosowane zarówno do problemów regresji (aproksymacji) jak i klasyfikacji poprzez nauczanie nadzorowane.

Podstawową składową każdej sieci neuronowej jest neuron, który może być reprezentowany jako funkcja wielu parametrów zwracająca dyskretną wartość prawda lub fałsz. Choć jeden neuron nie jest w stanie opisać złożonych modeli, dużo większe możliwości mają sieci neuronowe zbudowane z wielu neuronów połączonych w warstwy w taki sposób, że wynik jednej warstwy służy jako parametry kolejnej.

Poprzez odpowiednie ustawienie współczynników funkcji w neuronach jesteśmy w stanie zamodelować zależności dużo bardziej złożone niż te możliwe do opisania jedną prostą funkcją wielu parametrów.

2.2.1 Wielowarstwowy perceptron

W praktyce w sieciach neuronowych neurony są zastąpione perceptronami, które od neuronów różnią się tym, że są funkcjami w dziedzinie rzeczywistej. Same warstwy neuronów również nie są reprezentowane jako zbiory obiektów typu neuron. Operacja przechodzenia danych wejściowych przez kolejne warstwy sieci zwana propagacją w przód może być zrealizowana w każdej warstwie poprzez proste mnożenie macierzy

$$X \times W = S$$

Gdzie: X jest macierzą $(n, 1)$ parametrów będących danymi wejściowymi sieci w przypadku warstwy pierwszej (wejściowej), lub wartości zwróconych przez poprzednią warstwę w sieci

W jest macierzą (m, n) współczynników funkcji opisujących perceptrony, w której każdy wiersz opisuje jeden perceptron, a każda kolumna odpowiada wartości współczynnika tego perceptronu dla danego parametru z wektora wejściowego warstwy

S jest macierzą wynikową warstwy $(1, m)$ zawierającą wartości zwrócone przez funkcje opisujące perceptrony

tworzące tę warstwę.

Następnie należy zdecydować kiedy uznamy dany neuron za pobudzony. Do tego zadania stosuje się funkcje aktywacji, których dobór jest niezwykle ważny i może łatwo zadecydować o użyteczności sieci.

$$f(S) = Z$$

Gdzie:

$f(\cdot)$ jest wybraną funkcją aktywacji, która powinna być różniczkowalna by możliwe było użycie jej w propagacji wstecz,

S jest macierzą wynikową $(1, m)$ warstwy,

Z jest macierzą $(1, m)$, w której każdy element jest intensywnością pobudzenia danego neuronu.

Aby korzystać z sieci neuronowych do opisywania skomplikowanych modeli statystycznych nie możemy ręcznie decydować o wartościach parametrów w warstwach. Byłoby to niezwykle ciężkie o ile nie niemożliwe w sposób analityczny. Zamiast tego sieci neuronowe poddaje się trenowaniu.

Proces trenowania sieci neuronowej nazywany jest propagacją wstecz. Gdy przeprowadzimy proces propagacji w przód dla sieci z wartościami współczynników o wartościach losowych o dowolnym rozkładzie jesteśmy w stanie poprawić je stosując wybraną różniczkowalną funkcję straty i znając wartości pożądane. Dzieje się to w najprostrzym przypadku poprzez metodę spadku po gradiencie, którą można opisać macierzowo dla całej warstwy neuronów jako:

$$W_1 = W_0 - \eta \frac{\partial E}{\partial S}$$

Gdzie:

W_0 jest macierzą (m, n) współczynników perceptronów warstwy, a W_1 jest jej nową postacią

η jest współczynnikiem *learningrate* kontrolującym tempo uczenia

$\frac{\partial E}{\partial Z}$ jest pochodną z sygnału błędu po macierzy wynikowej S danej warstwy. Sygnał błędu w przypadku ostatniej warstwy (wyjściowej) jest gradientem funkcji błędu macierzy wynikowej, w przypadku reszty warstw jest on sygnałem błędu pochodzącym z poprzednio aktualizowanej w procesie propagacji wstecz warstwy wyznaczonym poprzez:

$$E_1 = - \frac{\partial E_0}{\partial S}$$

Gdzie:

E_1 jest sygnałem błędu zwracany przez daną warstwę,

$\frac{\partial E}{\partial S}$ jest pochodną cząstkową z sygnału błędu, który otrzymała ta warstwa od warstwy poprzedniej w procesie propagacji wstecz, lub w przypadku warstwy wyjściowej gradientem funkcji straty dla macierzy wyjściowej tej warstwy

2.2.2 Wielowarstwowy perceptron, własna implementacja

Z wykorzystaniem biblioteki numpy zaimplementowano model sieci neuronowej typu wielowarstwowy perceptron zgodny z powyższą teorią do bazy MNIST-784 zawierającej 70000 obrazów cyfr z przedziału 0-9. Architektura warstw stworzonej sieci wyglądała następująco:

$$784 \rightarrow 512, \text{sigmoid} \rightarrow 256, \text{sigmoid} \rightarrow 128, \text{sigmoid} \rightarrow 10, \text{softmax}$$

Za funkcję straty przyjęto funkcję Categorical Crossentropy, learning rate wynosił 0.01. W trakcie trenowania nie korzystano żadnych mechanizmów nieomówionych w części teoretycznej w tym: batchingu czy przetwarzania danych wejściowych poza skalowaniem do zakresu $[0, 1]$.

Ze zbioru MNIST-784 jako zbiór treningowy obrano 80% danych a jako zbiór testowy pozostałe 20%. Wartość accuracy predykcji tego modelu na zbiorze testowym osiągnęła pik w pobliżu 0.92 po sześciu epochach treningu.

2.2.3 Wielowarstwowy perceptron, tensorflow keras

Z wykorzystaniem pakietu tensorflow.keras utworzono sieć neuronową o architekturze analogicznej do tej zaimplementowanej ręcznie. Zastosowano kilka zmian w postaci:

Klasa	Precyzja	Zwrot	F1
1	0.95	0.97	0.96
2	0.93	0.97	0.95
3	0.90	0.91	0.92
4	0.95	0.89	0.92
5	0.87	0.92	0.9
6	0.92	0.96	0.94
7	0.91	0.87	0.89
8	0.9	0.87	0.88
9	0.9	0.92	0.9
0	0.95	0.97	0.96

Klasa	Precyzja	Zwrot	F1	Celność
1	0.91	0.99	0.95	0.98
2	0.92	0.97	0.94	0.99
3	0.85	0.96	0.90	0.98
4	0.97	0.92	0.94	0.99
5	0.98	0.85	0.91	0.98
6	0.97	0.94	0.95	0.99
7	0.95	0.94	0.95	0.98
8	0.97	0.9	0.93	0.98
9	0.93	0.92	0.93	0.98
0	0.97	0.97	0.97	0.99

- Do propagacji wstecz zamiast stochastycznego schodzenia po gradiencie użyto algorytmu Adam będącego jego udoskonaleniem uwzględniającym momenty gradientu.
- Zastosowano mechanizm batchingu, $batch - size = 128$

2.2.4 Sieć konwolucyjna

Sieć konwolucyjna jest odmianą sieci neuronowej, w której stosuje się warstwy konwolucyjne. W takich warstwach każdy neuron zamiast nakładać na zbiór parametrów funkcję, nakłada na nie filtr, którego wagi są współczynnikami neuronu. Filtr może być w postaci wektora, małej macierzy kwadratowej, lub kilku macierzy w zależności od tego czy interpretujemy dane wejściowe jako wektor, powierzchnię czy przestrzeń punktów. Odpowiednio wytrenowana sieć konwolucyjna jest w stanie skutecznie wykrywać i wzmacniać poprzez nakładanie filtrów istotne cechy danych, które następnie mogą posłużyć jako wejście dla warstwy zwykłych perceptronów, które dzięki temu wyszczególnieniu kluczowych cech dużo skuteczniej poradzą sobie w rozwiązaniu zadania.

Poza warstwami konwolucyjnymi w sieciach konwolucyjnych stosuje się często Max Pooling. Nie jest to warstwa neuronowa, lecz procedura przetwarzająca obrazy utworzone przez warstwy konwolucyjne zmniejszająca rozmiar tworzonych przez sieć obrazów poprzez wstawianie w miejsce każdego okna $n \times n$ wartość maksymalną z tego okna.

Kolejne udoskonalenie modelu polegało na dodaniu procedury Dropout jako jednej z warstw sieci. Dropout ustawia z pewnym prawdopodobieństwem pojedyncze wartości mu przekazane na 0 w ten sposób wykluczając pewne obserwacje sieci z rezultatu i zmniejszając szansę na przeuczenie sieci, co skutkowałoby doskonałym klasyfikowaniem danych, na których model był uczony, lecz nie radzeniem sobie zupełnie z nowymi nie widzianymi przez niego danymi.

Sieć konwolucyjną zaimplementowano przy użyciu pakietu tensorflow keras. Wszystkie hiperparametry pozostały identyczne z wielowarstwowym perceptronem zaimplementowanym z użyciem tych samych bibliotek. Struktura sieci wyglądała następująco:

$28x28 \rightarrow 32x(3,3), relu \rightarrow maxPooling(2,2) 64x(3,3), relu \rightarrow maxPooling(2,2) \rightarrow Flatten \rightarrow Dropout(0.5) \rightarrow 10, softmax$

Sieć konwolucyjna osiągnęła najwyższą celność na zbiorze testowym równą 0.989 po sześciu epokach nie wykazując dużych zmian przy dalszym trenowaniu. Jest to wynik dużo lepszy niż w przypadku zwykłego wielowarstwowego perceptronu. Dodatkowo proces uczenia tej sieci przebiegał o wiele szybciej niż wielowarstwowego perceptronu dzięki warstwom konwolucyjnym, które ograniczyły ilość potrzebnych warstw gęstych do jednej.

Klasa	Precyzja	Zwrot	F1
1	0.99	0.998	0.994
2	0.984	0.987	0.986
3	0.99	0.988	0.988
4	0.992	0.99	0.99
5	0.99	0.99	0.99
6	0.99	0.994	0.99
7	0.993	0.985	0.988
8	0.987	0.982	0.984
9	0.984	0.985	0.984
0	0.992	0.995	0.993

2.2.5 Sieć grafowa

2.2.6 Konwolucyjna sieć transferowa

Z racji, że różne zadania klasyfikacji obrazów nie różnią się od siebie tak drastycznie jak mogłoby się wydawać, popularnym sposobem na tworzenie bardzo dokładnych klasyfikatorów jest korzystanie z sieci transferowych. Są to sieci konwolucyjne korzystające z wytrenowanej już wcześniej na innych problemach klasyfikacji warstw konwolucyjnych. Proces nauczania takiej sieci neuronowej jest o wiele prostszy, gdyż wytrenowania wymaga jedynie mała ilość nowych warstw, które zinterpretują wyniki już gotowych i wytrenowanych w klasyfikacji obrazów warstw konwolucyjnych. W pierwszych etapach nauczania należy pominąć już wytrenowane warstwy konwolucyjne aby usprawnić ten proces. Gdy model jest już dostatecznie dobrze wytrenowany można odblokować trenowanie warstw konwolucyjnych, by przeprowadzić tzw. fine tuning, który pozwoli wyspecjalizować warstwy konwolucyjne w detekcji cech charakterystycznych dla danego problemu.

Sieć konwolucyjną transferową zbudowano z użyciem biblioteki tensorflow keras oraz modelu transferowego ResNet50 klasyfikującego obrazy w przestrzeni RGB o rozmiarze przynajmniej 75×75 px.

Z powodu rozbieżności rozmiarów obrazów, które wymaga sieć ResNet50 i obrazów należących do baz danych MNIST i podobnych konieczny był preprocessing danych wejściowych. Do zastosowanych metod należały:

- skalowanie obrazów 28×28 px do rozmiaru 75×75 px
- zklonowanie skali szarości na kanały RGB

Dodatkowo by uzyskać jak najlepsze rezultaty na danych nie tylko należących do bazy danych, ale stworzyć model, który spełni zadanie klasyfikacji możliwie tak dobrze jak człowiek zastosowano generowanie nowych danych treningowych w oparciu o dane z bazy z wykorzystaniem:

- skalowania zawartości obrazu
- obracania obrazów
- przesuwania obrazów w osiach X i Y

2.3 KNN - K Nearest Neighbours

KNN jest modelem bezparametrycznego uczenia nadzorowanego. Jest to niezwykle prosty model, który nie wymaga procesu uczenia, jednak przypląca to bardzo kosztowną predykcją. Algorytm K Nearest Neighbours polega na obliczeniu odległości wektora danych wejściowych długości n traktowanego jako punkt w przestrzeni n wymiarowej i porównaniu go z każdym z pośród wektorów danych wejściowych dostarczonych modelowi w ramach danych treningowych, które to model zapamiętał. Punkty z pośród danych treningowych wraz z ich etykietami następnie są sortowane rosnąco wedle ich odległości od danej wejściowej, której klasyfikację przeprowadza model. Następnie dopierane jest K pierwszych z pośród punktów, których to odległość od klasyfikowanego punktu jest najniższa i zliczane są wystąpienia różnych typów etykiet pośród wybranych K punktów. Ta etykieta, która pośród nich powtarza się najczęściej jest odpowiedzią modelu na zadanie klasyfikacji.

Model KNN został zaimplementowany z użyciem biblioteki numpy. Za parametr K wybrano wielokrotność liczności zbioru etykiet, 20.

Klasa	Precyzja	Zwrot	F1
1	0.92	1.0	0.96
2	0.97	0.91	0.94
3	0.98	0.95	0.96
4	0.96	0.96	0.96
5	0.96	0.98	0.97
6	0.97	0.99	0.98
7	0.93	0.94	0.93
8	0.98	0.90	0.94
9	0.96	0.95	0.96
0	0.98	0.99	0.99

Z powodu długiego czasu predykcji modelu KNN testy wykonano na 10% zbioru testowego, co odpowiada 1400 danych testowych. W tej sytuacji model KNN cechował się accuracy na poziomie 0.96

Macierz konfuzji

Z powodu swojej konstrukcji model KNN sprawował się niemal perfekcyjnie dla zbioru danych testowych, które przypominały w znacznym stopniu dane treningowe. Niestety dla ręcznych rysunków nienależących do bazy MNIST-784 model działał dużo gorzej od sieci konwolucyjnej, która nie tylko porównuje punkt z danymi treningowymi, lecz wynajduje schematy i cechy obrazów, które decydują o przynależności do danej klasy. Należy również wspomnieć, że z powodu nieporównywalnie większej ilości obliczeń koniecznych do wykonania przez model KNN w porównaniu do dowolnego innego modelu, podczas gdy proces weryfikacji na 14000 danych testowych sieci neuronowych trwał zaledwie kilka sekund, weryfikacja modelu KNN zajęła ponad 10 minut dla 1400 danych.

3 Wyniki

Natomiast tutaj będą bardzo ładne wyniki.

4 Wnioski

Najprawdopodobniej albo... albo szyny, które nie były... nie są równe, albo po prostu no, tak jak mówiłam we wcześniejszym wejściu, ee, yy, szyny... szyny były złe... a podwozie... podwozie... podwozie... podwozie też było złe. [1]

Źródła

- [1] Wikipedia, *Szyny były złe*
https://pl.wikipedia.org/wiki/Szyny_by%C5%82y_z%C5%82e
- [2] Super strona, *Jest super*
<https://www.example.com>
- [3] Marek Kubale, *Łagodne wprowadzenie do algorytmów*, 2022