Dumitru Florentin Giuliano

Grupa 233

# Documentatie

Am incarcat 14 solutii, folosind pentru primele 3 knn cu numar variat de vecini, iar pentru restul solutiilor am folosit neural network si cnn.

Citirea pozelor am facut-o cu ajutorul bibliotecii PIL(pillow) si cu ajutorul functiei nupmy.asarray() am convertit pozele citite in narray-uri pentru a putea sa le prelucrez in ambele situatii.

Codul pentru Knn este similar cu cel de la laborator. Mai exact am facut eu o clasa knnClasifier ce are 2 date membre, train_data, train_labels si o functie de predict ce primeste ca parametru o data si imi va prezice carui label ii apartine.

Predictia am facut astfel, am calculat distanta 2 pentru vectorii mei, ce au fost reshape-uiti de la 50x50 la 2500 si dupa am verificat care sunt cei mai aproape k vecini de mine. Dupa ce extrageam acesti vecini returnam label-ul ce apare cel mai frecvent.

In functie de numarul de vecini pe care i-am dat ca parametru functiei mele am observat ca variaza foarte mult acuratetea:

(Voi scrie acuratetea privata si dupa pe cea publica)

1) pentru 3 vecini : 0.45538 - 0.45128

2) pentru 10 vecini: 0.42461 - 0.42153

3) pentru 5 vecini: 0.48786 - 0.46666

Dupa am folosit un model de neural network bazat pe 2 layer-uri ascunse pe care l-am variat mereu din punct de vedere al numarului de neuroni folosit pe layer.

Modelul era unul simplu facut:

Am folosit din tensorflow.keras urmatoarele obiecte:

a) Model - ce reprezinta fix modelul meu

b) Dense - layer-ul de neuroni complet legat unul de altul

c) Input - un layer ce imi memoreaza dimensiunea datelor de intrare(2500).

d) Dropout - un layer ce ma ajuta sa previn overfit-ul, setand 0 pe anumite unitati de input cu o rata de frecventa data ca parametru. In cazul modelului meu am tot incercat sa modific aceasta rata, cea mai buna forma a modelului am avut-o cand rata era de 0.5 si Dense - urile mele de 64 si 32, de aceea am lasat in fisierul nn.py sub forma aceasta.

Alti parametri pe care i-am folosit in acest algoritm au fost functiile neliniare:

a) relu (ce imi va afla valoarea maxima dintre 0 si valoarea functiei)

b) softmax (ce imi transforma outputurile mele finale de pe ultimul layer in niste probabilitati pentru a putea sa spun ce predictie are modelul).

Datele mele sunt returanate sub forma unui narray de dim 3, ce are ca valoari acele probabilitati de asemanare cu fiecare dintre label-uri. Pentru a extrage ce valoare are predictia mea, am folosit functia numpy.argmax() ce imi returneaza indicele valorii cu cea mai mare probabilitate.

Parametrii de la functia de compile sunt:

a) functia de optimizare, eu am folosit adam, deoarece Adam calculeaza

gradientul se bazează pe estimarea adaptativă a momentelor de ordinul întâi și de ordinul al doilea.

b) functia de loss, ce ajuta la prevenirea overfitului. Aici am folosit SparseCategoricalAccuracy, deoarece am 3 label-uri posibile ca predictie.

c) ca metrici utilizate a fost acuratetea antrenarii, deoarece voiam sa vad in timp real cum evolueaza modelul.

Pentru functia de fit am folosit urmatorii parametrii:

a) train_foto, pozele pe care le avem si cu care antrenez modelul

b) train_Y_one_hot, acest parametru de fapt este un vector ce imi transforma array simplu de antrenare din (15000,) intr-unul de (15000,3) pentru a putea face fitul dintre label si neuronul de pe ultimul layer.

c) batch_size reprezinta numarul de date ce vor fi luate intr-un pas de antrenare.

d) epoch reprezinta numarul de pasi de antrenare pe care ii fac.

Din pacate pentru acest model nu am obtinut cele mai bune rate, dar am remediat acest lucru folosind dupa convutional neural network.

Pentru acest model am obtinut astfel, in functie de variatia numarului de neuroni de pe cele 2 hidden layers si a frecventei de dropout:

(Voi scrie acuratetea privata si dupa pe cea publica)

1) pentru un model cu 32 si 32 de neuronu si fata dropout:

0.34632 - 0.33641

2) pentru 16 si 16 neuroni si fara dropout: 0.35931 - 0.38358

3) Am avut eroare,deoarece apasasem submit pana sa se incarce fisierul cu predictii

4) pentru 8 si 8 si fara dropout: 0.32136 - 0.32307

5) am rulat tot pentru 8 si 8 dar am introdus un dropout prea mare de 0.8 si am obtinut:

0.31521 - 0.30051

Ulimul model folosit de mine in proiect este Convolutional Neural Network, ce mi-a oferit cea mai buna predictie.

In mare parte acest model este foarte similar cu Neural Network, doar ca am 4 obiecte noi adaugate ce imi face acea cautare de feature-uri dintre poze pentru am imi face o predictie cat mai exacta.

Cele 4 clase adaugate sunt:

a) Conv2D - aceasta clasa imi extrage din poze "subpoze" de dimensiune specificata si dupa le trece prin functia de pooling pentru a le vedea feature-urile.

b) AveragePooling2D - aceasta clasa imi face media valorilor primite de dimensiune specificata din Conv2D, si aici setam dimensiunea dreptunghiului pentru care se face aceasta medie.

c) MaxPool2D - Spre deosebire de layer-ul anterior, aceasta clasa nu-mi face media ci imi extrage valoarea maxima.

d) Flatten - imi transforma matricea de forma (nr_de_poze, 50,50,1) intr-una de froma (nr_de_poze,2500).

Pentru ambele modele pe care le-am folosit aici am pastrat aceasi structura pe care am pus-o in documetatie in codul cnn.py, diferenta o face ca am ales la functia de MaxPool2D ,la Conv2D si la numarul de neuroni de pe zonele ascunse valori diferite, lucru ce mi-a alterat valoarea ratei de predictie.

In rest, clasele, hyperparametrii si functiile sunt aceleasi folosite ca in modelul simplu de neural network, doar ca spre deosebire de el, aici am folosit si functia keras.utils.normalize, pentru a imi normaliza datele.

Predictiile in functie de aceste date sunt:

(Voi scrie acuratetea privata si dupa pe cea publica)

1) pentru un model cu 32 de neuroni , 32 de neuroni si un dropout de 0.5 am obtinut si dimensiunea de 5x5 a Con2D si 2x2 la AvragePool2D:

0.42188 - 0.40820

2) pentru acelasi model am mai rulat si am obtinut:

0.42803 - 0.39487

3) am scazut numarul de neuroni la 16 si 16, iar dropoutul l-am facut de 0.6 si am obtinut si dimensiunea de 5x5 a Con2D si 2x2 la AvragePool2D:

0.34495 - 0.37230

4) 64 de neuroni pe primul layer si 32 pe al doilea, iar frecventa de dropout dintre ele este de 0.5 si dimensiunea de 3x3 a primului Con2D si 5x5 al celui de-al doilea Conv2D si 2x2 la AvragePool2D:

0.43487 - 0.42461

5) pentru dimensiune 3x3 la ambele Conv2D si AvragePool2D,iar MaxPool2D este de 3x3 : 0.50188 - 0.49230

6) pentru dimensiune 2x2 la AvragePool2D , dar restul nemoddificate: 0.47282 - 0.46974

Modeulul meu de cnn a extras prima data feature -uri de dimensiune 3x3 din pozele initiale si dupa tot 3x3 din cele obtinute, deoarece am cautat asemanari intre ele in functie de tipul lor.

Am facut matricea de confuzie pentru modelele finale luate de mine, mai exact cele ce mi-au dat acuratetea cea mai buna pentru fieacare dintre cazuri.

Voi explica forma tabelelor acum:

a) Linia reprezinta valoarea ce trebuiau prezise.

b) coloana reprezinta valoarea prezisa de fapt.

1) Matricea de confuzie pentru knn:

| | 0 | 2 | 3 |
|---|---|---|---|
| 0 | 854 | 264 | 382 |
| 1 | 309 | 672 | 519 |
| 2 | 315 | 502 | 683 |

2) Matricea de confuzie pentru nn:

| | 0 | 2 | 3 |
|---|---|---|---|
| 0 | 514 | 776 | 210 |
| 1 | 494 | 787 | 219 |
| 2 | 531 | 766 | 203 |

3) Matricea pentru cnn:

| | 0 | 2 | 3 |
|---|---|---|---|
| 0 | 772 | 410 | 318 |
| 1 | 469 | 605 | 426 |
| 2 | 391 | 312 | 797 |

Pentru a calcula acuratetea, adunam valorile de pe coloana principala si le impartim la suma tuturor valorilor.