[WSI] Laboratorium 5 – MLP

Wykonali: Andrii Gamalii, numer indeksu: 323665

Szymon Jankowski, numer indeksu: 318662

O programie

Python wersji 3.10.0 64-bit

W rozwiązaniu użyliśmy poniższych bibliotek zewnętrznych:

- pygame~=2.1.0
- torch~=1.13.0+cu116
- numpy~=1.23.5
- scikit-learn~=1.1.3
- torchmetrics~=0.11.0

Atrybuty

Zdefiniowaliśmy 10 atrybutów:

1. obecny kierunek
2. długość węża (w blokach)
3. odległość od głowy do górnej ściany
4. odległość od głowy do dolnej ściany
5. odległość od głowy do lewej ściany
6. odległość od głowy do prawej ściany
7. czy jest jedzenie na prawo od głowy (tak = 2, nie = -2)
8. czy jest jedzenie na lewo od głowy (tak = 2, nie = -2)
9. czy jest jedzenie na górze od głowy (tak = 2, nie = -2)
10. czy jest jedzenie na dole od głowy (tak = 2, nie = -2)

Miara dokładności

Wzięliśmy zbiór danych długości 3452, z niego zbiory treningowy, walidujący i testowy w proporcji 8:1:1 i zmierzyliśmy miarę dokładności modeli z learning_rate = 0,01 batch_size = 20 oraz n_epochs = 5. Uzyskaliśmy takie wyniki:

nn.Identity()	
Ilość warstw ukrytych	Dokładność na zbiorze testowym
1	0,95
2	0,89
5	0,84
30	0,33

nn.ReLu()	
Ilość warstw ukrytych	Dokładność na zbiorze testowym
1	0,93
2	0,81
5	0,33
30	0,33

nn.LeakyReLu(0,01)	
Ilość warstw ukrytych	Dokładność na zbiorze testowym
1	0,90
2	0,85
5	0,33
30	0,33

nn.Tanh()	
Ilość warstw ukrytych	Dokładność na zbiorze testowym
1	0,90
2	0,93
5	0,57
30	0,33

Dla tych hiperparametrów możemy zaobserwować pewne zależności. Da się zauważyć, że najlepsze wyniki model osiąga dla małej ilości warstw ukrytych. Najczęściej najlepsza dokładność jest dla tylko jednej ukrytej warstwy. Nie licząc funkcji aktywacyjnej nn.Identity przy 5 ukrytych warstwach widać duży spadek dokładności. Przy 30 warstwach ukrytych dokładność jest na bardzo niskim poziomie i zawsze osiąga taką samą wartość.

Średnia norma gradientów w czasie pierwszej epoki trenowania dla 30 warstw ukrytych i funkcji aktywacyjnej ReLU :

['0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000000', '0.000002', '0.000004', '0.000013', '0.000013', '0.000143', '0.000594', '0.001632', '0.004192', '0.006134', '0.027005', '0.061417', '0.304014']

Im dalsza warstwa tym średnia norma gradientów jest większa.

Wpływ liczby neuronów w warstwie ukrytej

Dla 1 warstwy ukrytej oraz funkcji aktywacji ReLU wyniki dokładności dla różnych liczb neuronów na warstwie ukytej.

Liczba neuronów na ukrytej warstwie	Dokładność na zbiorze testowym	Dokładność na zbiorze treningowym
10	0,91	0,91
8	0,83	0,91
6	0,77	0,89
4	0,72	0,83

Im więcej neuronów w warstwie ukrytej tym wyższa dokładność na zbiorze testowym naszego modelu. Na mniejszej liczbie neuronów w warstwie ukrytej możemy zaobserwować przeuczenie. Dokładność na zbiorze testowym jest niższa od tej na zbiorze treningowym.

By przeciwdziałać przeuczeniu użyliśmy metody Dropout(p=0.3), która zeruje pewne wartości na wejściu. Poniżej w tabelce przedstawione są wyniki po użyciu tej metody:

Liczba neuronów na ukrytej warstwie	Dokładność na zbiorze testowym	Dokładność na zbiorze treningowym
10	0,81	0,69
8	0,75	0,70
6	0,88	0,73
4	0,84	0,70

Można zaobserwować, że przeuczenie zostało wyeliminowane. Dokładność na zbiorze testowym jest nawet wyższa od tej na zbiorze treningowym.

Teraz w celu wyeliminowania przeuczenia użyliśmy regularyzacji L2. Poniżej można zaobserwować wyniki:

Liczba neuronów na ukrytej warstwie	Dokładność na zbiorze testowym	Dokładność na zbiorze treningowym
10	0,92	0,90
8	0,89	0,81
6	0,91	0,87
4	0,73	0,73

Tutaj przeuczenie także zostało wyeliminowane, a wyniki są nawet lepsze niż w przypadku użycia metody Dropout().

Agent MLP

W agencie MLP tworząc model użyliśmy następujących parametrów: funkcja aktywacyjna=nn.Identity(), liczba warstw ukrytych=1, liczba neuronów na warstwie ukrytej=10, liczba epok=10. Taki model dawał najlepsze rezultaty. Dla 100 gier efekt był taki:

- Maksymalny wynik: 30
- Średni wynik 14,47

Dokładność tego modelu na zbiorze testowym wynosiła: 0,94.

Log trenowania tego modelu wyglądał następująco:

{

```
 "0": {
  "train_accuracy": 0.792805552482605,
  "validate_accuracy": 0.8144927620887756,
  "loss": 88.0348673760891
 },
 "1": {
  "train_accuracy": 0.8863309025764465,
  "validate_accuracy": 0.8376811742782593,
  "loss": 52.07243192195892
 },
 "2": {
  "train_accuracy": 0.9169062972068787,
  "validate_accuracy": 0.9188405871391296,
  "loss": 43.543020851910114
 },
 "3": {
  "train_accuracy": 0.9237407445907593,
  "validate_accuracy": 0.8608695864677429,
  "loss": 39.83292520046234
 },
 "4": {
  "train_accuracy": 0.9341722726821899,
  "validate_accuracy": 0.9159420132637024,
  "loss": 37.0192166864872
 },
 "5": {
  "train_accuracy": 0.939568042755127,
  "validate_accuracy": 0.9420289993286133,
  "loss": 35.3776388540864
 },
 "6": {
  "train_accuracy": 0.9420859813690186,
  "validate_accuracy": 0.8666666746139526,
  "loss": 34.590214766561985
 },
 "7": {
  "train_accuracy": 0.9438846707344055,
  "validate_accuracy": 0.9362319111824036,
  "loss": 33.9321077009663
 },
 "8": {
  "train_accuracy": 0.9507191777229309,
  "validate_accuracy": 0.9420289993286133,
  "loss": 32.783869575709105
 },
 "9": {
  "train_accuracy": 0.9496398568153381,
  "validate_accuracy": 0.947826087474823,
  "loss": 32.101624853909016
 }
}
```

Agent wykorzystujący SVM, trenowany na innym zbiorze danych, osiągał wyniki około 3. Model wykorzystujący sieci neuronowe jest w stanie uzyskać dużo lepszy wynik.