**8. Algorytmy sztucznych sieci neuronowych**

Sieci neuronowe można uważać za doskonałe wytwory uczenia maszynowego. Stanowią sposób mapowania wejść i wyjść. Wspaniałe w sieciach neuronowych jest to, że (w odróżnieniu od perceptronów) mogą być używane do modelowania złożoności. Jeśli na przykład mamy trzy wejścia i jedno wyjście, moglibyśmy arbitralnie ustawić wewnętrzną złożoność na 2 lub 3 jedynie w oparciu o naszą wiedzę z dzieciny danego problemu.

**8.1. Perceptron**

- konstrukcja najprostszej sztucznej sieci neuronowej Zadaniem perceptronu jest klasyfikacja podanego na wejście wektora *x* do jednej z dwóch klas *C1* lub *C2*. Jeśli sygnał wyjściowy neuronu przyjmuje wartość 11 to wektor xx jest zaklasyfikowany do klasy *C1*, jeśli przyjmuje wartość *0* to do klasy *C2*. Perceptron dzieli *N*-wymiarową przestrzeń wektorów wejściowych na dwie półprzestrzenie rozdzielone *N-1* wymiarową hiperpłaszczyzną. Hiperpłaszczyzna ta zwana jest granicą decyzyjną. Jeśli przestrzeń obiektów jest dwuwymiarowa to rozdzielająca je granica jest linią prostą.

**8.2. Propoagacja wsteczna (Back-Propagation)**

W przypadku sieci wielowarstwowych najczęściej stosowanym algorytmem uczenia jest *algorytm wstecznej propagacji błędów*. Jego nazwa pochodzi stąd, iż po obliczeniu odpowiedzi sieci na zadany wzorzec, obliczana jest wartość gradientu funkcji błędu dla neuronów ostatniej warstwy. Następnie modyfikuje się ich wagi. Błąd jest propagowany do warstwy wcześniejszej (przedostatniej). Wartości funkcji gradientu dla neuronów z tej warstwy obliczane są w oparciu o gradienty dla neuronów z warstwy następnej (czyli ostatniej). Modyfikowane są wagi kolejnej warstwy. Postępowanie trwa aż do warstwy wejściowej.

Algorytm wstecznej propagacji błędów:

• Podanie na wejście wektora uczącego x i obliczenie aktualnego wyjścia y

• Obliczenie błędów w warstwie ostatniej na podstawie różnicy otrzymanych wartości

wektora y oraz wzorcowych d

• Adaptacja wag od warstwy wejściowej do wyjściowej

• Obliczenie błędów dla neuronów we wszystkich warstwach wcześniejszych po kolei

(jako funkcji błędu warstwy następnej, który jest już znany)

• Powtarzamy procedurę do momentu kiedy sygnały wyjściowe sieci będą dostatecznie

bliskie oczekiwanym

Istnieje wiele wariantów i ulepszeń oryginalnej metody wstecznej propagacji błędów. Pozwalają one uzyskać lepsze efekty uczenia lub przyśpieszyć proces nauki.

KOD:

MLP (Multi-Layer Percetron) trenuje z wykorzystaniem Backpropagation. Dokładniej, trenuje przy użyciu jakiejś formy descent gradient, a gradienty są obliczane przy użyciu propagacji wstecznej.

**from** sklearn.neural\_network **import** MLPClassifier

X = [[0., 0.], [1., 1.]]

y = [0, 1]

clf = MLPClassifier(solver=**'lbfgs'**, alpha=1e-5, hidden\_layer\_sizes=(5, 2), random\_state=1)

clf.fit(X, y)

*# MLPClassifier(activation='relu', alpha=1e-05, batch\_size='auto',*

|  |  |
| --- | --- |
|  | *beta\_1=0.9, beta\_2=0.999, early\_stopping=False, epsilon=1e-08, hidden\_layer\_sizes=(5, 2), learning\_rate='constant', learning\_rate\_init=0.001, max\_iter=200, momentum=0.9, n\_iter\_no\_change=10, nesterovs\_momentum=True, power\_t=0.5, random\_state=1, shuffle=True, solver='lbfgs', tol=0.0001, validation\_fraction=0.1, verbose=False, warm\_start=False)* |

*# After fitting (training), the model can predict labels for new samples:* clf.predict([[2., 2.], [-1., -2.]]) *# array([1, 0])*

Parametry MPLClassifier ():

• **hidden\_layer\_sizes (tuple, length = n\_layers - 2, domyślnie (100,))**

– I-ty element reprezentuje liczbę neuronów w i-tej ukrytej warstwie.

**activation ({‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, domyślnie ‘relu’)**

- Funkcja aktywacji ukrytej warstwy.

„identity”, brak aktywacji, przydatne do implementacji linear bottleneck,, zwraca f (x) = x

„logistic”, logistyczna funkcja sigmoid, zwraca f (x) = 1 / (1 + exp (-x)). „tanh”, funkcja hiperboliczna tan, zwraca f (x) = tanh (x).

„relu”, funkcja rektyfikowanej jednostki liniowej, zwraca f (x) = max (0, x)

• **solver ({‘lbfgs’, ‘sgd’, ‘adam’}, domyślnie ‘adam’)**

- Solver do optymalizacji masy.

„lbfgs” to optymalizator w rodzinie metod quasi-Newton. „sgd” odnosi się do stochastycznego spadku gradientu.

„adam” odnosi się do stochastycznego optymalizatora opartego na gradiencie zaproponowanego przez Kingmę, Diederika i Jimmy'ego Ba

Uwaga: Domyślny solver „adam” działa całkiem dobrze na stosunkowo dużych zestawach danych (z tysiącami próbek szkoleniowych lub więcej) zarówno pod względem czasu szkolenia, jak i wyniku weryfikacji. Jednak w przypadku małych zestawów danych „lbfgs” może zbiegać się szybciej i działać lepiej.

• **alpha (float, domyślnie 0.0001**

- Parametr kary L2 (termin regularyzacji).

• **batch\_size (int lub ‘auto’)**

- Rozmiar minibatches dla stochastycznych optymalizatorów. Jeśli solver to „lbfgs”, klasyfikator nie będzie używał minibatchu. Po ustawieniu na „auto”, batch\_size = min (200, n\_samples).

• **learning\_rate ({‘constant’, ‘invscaling’, ‘adaptive’}, domyślnie ‘constant’)**

– Plan kursów nauki dla aktualizacji wagi.

„constant” to stała szybkość uczenia się podana przez „learning\_rate\_init”. „inyscaling” stopniowo zmniejsza szybkość uczenia się na każdym etapie „t”, stosując odwrotny wykładnik skalowania „power\_t”. efektywna\_nauczenie\_rate = learning\_rate\_init / pow (t, power\_t)

„adaptive” utrzymuje stałą częstotliwość uczenia się na poziomie „learning\_rate\_init”, o ile utrata treningu maleje. Za każdym razem, gdy dwie kolejne epoki nie zmniejszają utraty treningu o co najmniej tol lub nie zwiększają wyniku walidacji o co najmniej tol, jeśli włączone jest „wczesne zatrzymanie”, bieżąca szybkość uczenia się jest dzielona przez 5.

Używany tylko, gdy solver = 'sgd'..

• **learning\_rate\_init (double, domyślnie 0.001)**

- Zastosowana początkowa szybkość uczenia się. Kontroluje wielkość kroku w aktualizacji wag. Używany tylko wtedy, gdy solver = „sgd” lub „adam”..

• **power\_t (double, domyślnie 0.5)**

- Maksymalna liczba iteracji. Solver wykonuje iterację do konwergencji (określonej przez „tol”) lub tej liczby iteracji. W przypadku solverów stochastycznych („sgd”, „adam”) należy pamiętać, że określa to liczbę epok (ile razy zostanie użyty każdy punkt danych), a nie liczbę stopni gradientu.

• **shuffle (bool, domyślnie True)**

- Czy tasować próbki w każdej iteracji. Używany tylko wtedy, gdy solver = „sgd” lub „adam”.

• **random\_state (int, RandomState instance lub None, domyślnie None)**

• **tol (float, domyślnie 1e-4)**

- Tolerancja dla optymalizacji. Gdy utrata lub wynik nie poprawia się przynajmniej o tol dla kolejnych iteracji n\_iter\_no\_change, chyba że parametr learning\_rate jest ustawiony na „adaptive”, uznaje się, że konwergencja została osiągnięta i trening się kończy.

• **verbose (bool, domyślnie False)**

- Określa, czy drukować komunikaty o postępach na standardowe wyjście.

• **warm\_start (bool, domyślnie False)**

- Po ustawieniu wartości True ponownie użyj rozwiązania z poprzedniego połączenia, aby dopasować się do inicjalizacji, w przeciwnym razie po prostu usuń poprzednie rozwiązanie.

• **momentum (float, domyślnie 0.9)**

- Pęd do aktualizacji spadku gradientu. Powinien wynosić od 0 do 1. Używany tylko, gdy solver = „sgd”.

• **nesterovs\_momentum (bool, domyślnie True)**

- Czy użyć rozpędu Niestierowa. Używany tylko, gdy solver = ’sgd’ i momentum> 0

• **early\_stopping (bool, domyślnie False)**

- kreśla, czy zastosować wcześniejsze zatrzymanie, aby zakończyć trening, gdy wynik walidacji nie poprawia się. Jeśli ustawione na true, automatycznie odłoży 10% danych treningowych na walidację i zakończy trening, gdy wynik walidacji nie poprawi się przynajmniej o tol dla n\_iter\_no\_change kolejnych epok. Podział jest rozwarstwiony, z wyjątkiem ustawienia wielopłaszczyznowego. Działa tylko wtedy, gdy solver = „sgd” lub „adam”

• **validation\_fraction (float, domyślnie 0.1)**

- Odsetek danych treningowych, które należy odłożyć jako zestaw walidacyjny dla wczesnego zatrzymania. Musi zawierać się w przedziale od 0 do 1. Używany tylko, jeśli wczesne wyłączanie ma wartość True

• **beta\_1 (float, domyślnie 0.9)**

- Szybkość wykładniczego rozkładu dla szacunków wektora pierwszego momentu w adam powinna wynosić [0, 1). Używane tylko, gdy solver = „adam

• **beta\_2 (float, domyślnie 0.999)**

- Szybkość wykładniczego rozkładu dla oszacowań wektora drugiego momentu w adam powinna wynosić [0, 1). Używane tylko, gdy solver = „adam

Atrybuty:

• **classes\_ : array or list of array of shape (n\_classes,) -** Etykiety klas dla każdego

wyjścia

• **loss\_ : float -** Bieżąca strata obliczona za pomocą funkcji straty..

• **coefs\_ : list, length n\_layers – 1 -** I-ty element na liście reprezentuje macierz wag

odpowiadającą warstwie

• **intercepts\_ : list, length n\_layers – 1 -** I-ty element na liście reprezentuje wektor

odchylenia odpowiadający warstwie i + 1.

• **n\_iter\_ : int, -** Liczba iteracji uruchomionych przez solver.

• **n\_layers\_ : int -** Liczba warstw.

• **n\_outputs\_ : int -** Liczba wyjść.

• **out\_activation\_ : string -** Nazwa funkcji aktywacji wyjścia.

Metody:

• **fit (X, y)** – dopasuj model

• **get\_params()** – uzyskaj parametry estymatora

• **predict(X)** – przewiduj

• **predict\_proba(X)** – oszacowania prawdopodobieństwa

• **score(X, y[, sample\_weight])** – zwraca średnią dokładność danych testowych i etykiet.

• **set\_params(params)** – ustaw parametry estymatora