Paweł Put, Informatyka Stosowana gr. 3

Sprawozdanie nr 1.

Temat ćwiczenia: Budowa i działanie sieci jednowarstwowej.

Cel ćwiczenia: Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

Przebieg: Zaimplementowałem w javie dane do nauczania sieci w postaci 10 dużych i 10 małych liter polskiego alfabetu w wektorach 35cio elementowych, które po podzieleniu wyglądały jak tablice 5x7. Przykład dużej litery A:

* Wektor:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

* Tablica utworzona z wektora:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

* Interpretacja numeryczna wektora:

(0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1)

* Interpretacja numeryczna tabeli:

(0, 1, 1, 1, 0,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1 )

Następnie zaimplementowałem sieć jednowarstwową składającą się z 35 wejść (gdyż moje litery są wymiarów 5x7), 35 wag (po jednej dla każdego z wejść) i 1 wyjście, którego wartość wynosi 1, jeżeli litera jest duża, a 0, jeżeli mała. Pomysł na implementację wziąłem ze strony: <http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm>.

Algorytm nauczania:

1. Inicjujemy wagi losowo
2. Ustawiamy nauczanie sieci dla 20 neuronów o 35 wejściach odpowiadających literom alfabetu i dajemy im odpowiednie wartości wyjścia. Pierwsze 10 liter na wyjście otrzymuje 1, a kolejne 10 - 0.
3. Obliczamy wartość wyjścia dla wylosowanych wag i odchylenia standardowego.
4. Jeżeli wartość wyjścia i oczekiwanego wyjścia się zgadza, przechodzimy do następnego przypadku. W przeciwnym razie modyfikujemy wagi połączeń i odchylenie standardowe tak, by działały dla aktualnej pary neuronów i wracamy do pierwszego wypadku.
5. Powtarzamy krok 4 do momentu, aż przejdziemy przez wszystkie pary neuronów nie zmieniając wartości wag i odchylenia standardowego.

Wzór na modyfikowanie wag oraz odchylenia standardowego:

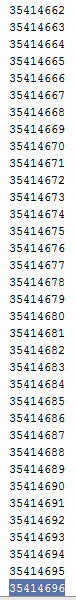
* weight[i] += learnCoefficient \* (expectAnswer - out) \* in[i]
* bias += learnCoefficient \* (expectAnswer - out),

gdzie:

* weight[i] - waga przy i-tym neuronie
* learnCoefficient - współczynnik nauczania podawany przez użytkownika
* expectAnswer - wartość oczekiwana dla danej pary neuronów
* out - wartość wyliczona ze wzoru
* in[i] - wartość i-tego neuronu
* bias - wartość odchylenia

Wartość wyjściową otrzymujemy dodając za każdym razem iloczyn wszystkich wag i neuronów, a następnie dodając wartość odchylenia standardowego. Jeżeli podana wartość jest większa lub równa 0 to na wyjściu otrzymujemy logiczną 1, w przeciwnym wypadku otrzymujemy 0.

Jako, że wykorzystuję algorytm z poprzedniego programu to znam jego optymalne wartości. Niestety przy użyciu tych wartości moja sieć jednowarstwowa nie nauczyła się rozróżniać wielkości liter po aż 35414696 epokach. Dorzucam screena z outputu z liczbą pętli:



W tym wypadku postanowiłem postawić na sieć wielowarstwową, zbudowaną w pythonie za pomocą biblioteki pybrain3. Jest to sieć jednokierunkowa, w której informacje przekazywane są w jednym kierunku, od wejścia do wyjścia. Moja sieć składa się z warstwy wejściowej składającej się z 35 wejść (gdyż mamy 35 wejść w każdej literze), 20 wyjść (gdyż mamy rozpoznać 20 różnych liter) i do tego 5 w warstwie ukrytej. Do tego połączyłem wszystkie warstwy w sposób "full connection" co oznacza, że każdy neuron z warstwy wejściowej połączony jest z każdym neuronem z warstwy ukrytej oraz że każdy neuron z warstwy ukrytej połączony jest z każdym neuronem z warstwy wyjściowej. Neurony warstwy ukrytej są sigmoidalne, co oznacza, że jego funkcja aktywacji przybiera postać funkcji sigmoidalnej, unipolarnej lub bipolarnej.

Po utworzeniu sieci zostało ją tylko wytrenować algorytmem wstecznej propagacji błędu. Jest to podstawowy algorytm uczenia nadzorowanego(z nauczycielem) wielowarstwowych sieci neuronowych. Ogólny schemat procesu trenowania sieci wygląda następująco:

1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo (na małe wartości).
3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią y oraz poprawną odpowiedzią t.
5. Błędy propagowane są do wcześniejszych warstw.
6. Każdy neuron (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkości przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3ciego dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
8. Zatrzymujemy się, gdy średni błąd na danych treningowych przestanie maleć. Możemy też co jakiś czas testować sieć na specjalnej puli nieużywanych do treningu próbek testowych i kończyć trenowanie, gdy błąd przestanie maleć.

Aby znaleźć taki zestaw wag, dla którego błąd sieci jest jak najmniejszy, możemy zapisać ten błąd

Funkcję od wartości wag. Zwykle błąd liczony jest jako kwadrat odchylenia: d = 1/2 (y-t)2.

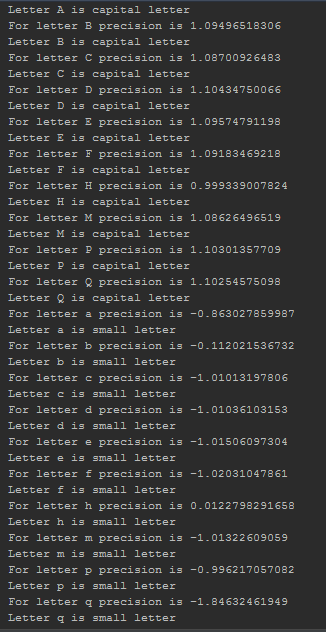
W tym wypadku będziemy operowali współczynnikiem uczenia. Współczynnik uczenia przyjmuje wartość z przedziału (0, 1>. Jest to wartość, przez którą zostanie pomnożona korekta wag. W programie ustawiane jest to w pliku training.py przy funkcji BackpropTrainer argument learningrate.

Przyjmiemy współczynniki uczenia:

* 0,01
* 0,1
* 0,5
* 0,8

|  |  |
| --- | --- |
| **Współczynnik uczenia** | **Liczba epok** |
| 0,01 | 1000 |
| 0,1 | 150 |
| 0,5 | 30 |
| 0,8 | 8 |

Screen outputu:



Wykres:

Omówienie wyników:

* Im większy współczynnik nauczania, tym szybciej uczyła się nasza sieć
* Podczas gdy jednowarstwowa sieć po 35414696 nie nauczyła się niczego, sieć z odpowiednim współczynnikiem nauczania i 5 neuronami w warstwie ukrytej potrzebowała 8 epok, by odróżnić, jakie litery jej prezentujemy

Wnioski:

* Należy odpowiednio wybrać sieć, którą zamierzamy nauczać. Nawet 5 neuronów w ukrytej warstwie zrobiło kolosalne zmiany w nauczaniu.
* Wraz ze wzrostem współczynnika nauczania szybciej uczyła się nasza sieć