**PODSTAWY SZTUCZNEJ INTELIGENCJI**

Scenariusz 3

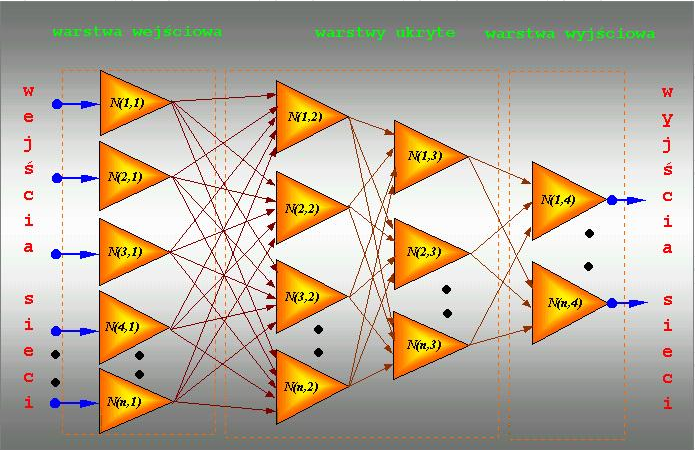
Maria Podkalicka, IS grupa 3

Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie z użyciem algorytmu wstecznej propagacji błędu rozpoznawania konkretnych liter alfabetu.

**Sieć wielowarstwowa:**

Jednokierunkowe, wielowarstwowe sieci neuronowe, czyli inaczej sieci typu MLP (ang. Multi-Layered Perceptron - wielowarstwowy perceptron), są najczęściej opisywaną i najchętniej wykorzystywaną w zastosowaniach praktycznych architekturą neuronową. Ich upowszechnienie związane jest z opracowaniem algorytmu wstecznej propagacji błędu (ang. Error Back Propagation), który umożliwił skuteczne trenowanie tego rodzaju sieci w stosunkowo prosty sposób. Oczywiście, od momentu opracowania, algorytm doczekał się licznych modyfikacji i ulepszeń, nie podważających jednak skuteczności pierwowzoru i stanowiących większe lub mniejsze modyfikacje jego podstawowej postaci.

Jej schemat jest następujący:



Wielowarstwowa sieć neuronowa jest w stanie przybliżać dowolnie złożone i skomplikowane odwzorowanie. Przy czym użytkownik nie musi znać lub zakładać z góry żadnej formy występujących w poszukiwanym modelu zależności, nie musi nawet zadawać sobie pytania czy jakiekolwiek możliwe do matematycznego modelowania zależności w ogóle występują. Cecha ta w połączeniu z samodzielną metoda uczenia się sieci neuronowej, czyni z niej niezwykle użyteczne i wygodne narzędzie do wszelkiego rodzaju zastosowań związanych z prognozowaniem, klasyfikacją lub automatycznym sterowaniem. Dodatkową zaletą jest duża tolerancja systemu neuronowego na uszkodzenia poszczególnych jej elementów, co podnosi niezawodność w sposób praktycznie niemożliwy do osiągnięcia w przypadku tradycyjnego oprogramowania, np. klasycznych systemów ekspertowych. Dodatkowo równoległy model przetwarzania sygnałów wejściowych i stosunkowo prosta struktura nauczonej sieci neuronowej pozwala osiągnąć bardzo szybki czas reakcji, pozwalający na wykorzystywanie jej w zadaniach wymagających reagowania w czasie rzeczywistym.

Projektowanie sieci neuronowej zaczyna się już na poziomie analizy sformułowanego problemu. Inaczej mówiąc, jakie i ile danych chcemy lub możemy podać na wejścia sieci (zdeterminuje to wielkość warstwy wejściowej) oraz jaką odpowiedź chcemy uzyskać ( ilość wyjść sieci). Pozostanie zatem do określenia ilość warstw ukrytych i neuronów w tych warstwach. Jest to natrudniejszy moment tego etapu pracy. Przyjmuje się, że sieć z jedną warstwą ukrytą powinna nauczyć się rozwiązywania większości postawionych problemów. Nie znane są problemy wymagające do rozwiązania sieci z więcej niż trzema warstwami ukrytymi. Nie ma natomiast dobrej recepty na dobór właściwej ilości neuronów w warstwie ukrytej. Można próbować według wzoru:



gdzie:

*Nwu - ilość neuronów w warstwie ukrytej*

*Nwwe - ilość neuronów w warstwie wejściowej*

*Nwwy - ilość neuronów w warstwie wyjściowej*

Generalnie jednak uczenie rozpoczyna się z małą ich ilością a następnie, obserwując postępy tego procesu, doświadczalnie zwiększa się ich ilość.

**Algorytm wstecznej propagacji błędu:**

Jest to podstawowy algorytm uczenia nadzorowanego wielowarstwowych, jednokierunkowych [sieci neuronowych](https://pl.wikipedia.org/wiki/Sie%C4%87_neuronowa). W tym algorytmie błąd ostatniej warstwy jest przesyłany wstecz i wykorzystywany do zmiany wartości wag w poprzednich warstwach.

**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 wielkosci przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. 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 znależć taki zestaw wag, dla którego błąd sieci jest jak najmniejszy, możemy zapisać ten błąd jako funkcję od wartości wag. Oznaczmy przez:

f: R -> R – funkcję aktywacji w neuronie

w1 ,..., wK – wagi połączeń wchodzących

z1 ,..., zK – sygnały napływające do neuronu z poprzedniej warstwy.

Zwykle błąd liczony jest jako kwadrat odchylenia: d = 1/2 (y-t)2, co możemy rozpisać jako:

**d(w1, ... , wk) = (f(w1z1+... + wkzk)-t)2**

W celu ustalenia, o ile powinna zmienić się waga neuronu, powiniśmy "rozłożyć" otrzymany błąd całkowity na połączenia wprowadzające sygnały do danego neuronu. Składową błędu dla każdego j-tego połączenia określamy jako pochodną cząstkową błędu względem j-tej wagi.

**Dane uczące i testujące:**

Przygotowany przeze mnie zestaw uczący zawiera 20 dużych liter alfabetu łacińskiego. Każda litera utworzona jest na matrycy 5x5 czyli na 25 polach.

Przykład : litera W

1 0 0 0 1  
1 0 0 0 1  
1 0 0 0 1  
1 0 0 0 1  
1 0 1 0 1  
1 0 1 0 1  
0 1 0 1 0  
Liniowym rozwinięciem tych matryc będą wektory wejściowe ciągu uczącego. Poniżej ciąg uczący składający się z 5 następujących wektorów uczących :

A= {1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1};

B= {1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0};

C= {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1};

D= {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

E= {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1};

**Wyniki:**

Poniższa tabela przedstawia wyniki w zależności od współczynnika uczenia

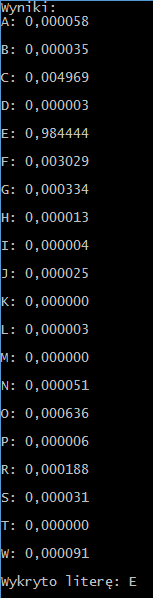
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Współczynnik uczenia** | 0,3 | 0,5 | 0,7 | 0,9 | 1 |
| **Wartość uzyskana dla wykrytej litery** | 0,975 | 0,981 | 0,984 | 0,986 | 0,995 |
| **Czas algorytmu [s]** | 1,6 | 1,2 | 0.9 | 0.8 | 0.7 |

Wykres przedstawia zależność wartości uzyskanej dla wykrytej litery od współczynnika uczenia.

Poniższa tabela przedstawia różne wartości współczynnika bezwładności oraz ilość potrzebnych iteracji dla danego współczynnika.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Wartość współczynnika bezwładności** | 0,5 | 0,8 | 0,9 | 1,1 |
| **Ilość potrzebnych iteracji** | 9999 | 7600 | 6700 | 8100 |

**Screen działającego programu dla współczynnika uczenia 0.9:**



**Analiza i dyskusja błędów uczenia i testowania opracowanej sieci w zależności od wartości współczynnika uczenia i bezwładności**

Im wyższy współczynnik uczenia, tym litery zostały szybciej i lepiej rozpoznane. Zbyt mały współczynnik uczenia powoduje wolne oraz mniej znacznie mniej efektywne rozpoznawanie oraz dłuższy czas rozpoznawania.

Druga tabela pokazuje, że współczynnik bezwładności nie może być za duży, ponieważ sieć będzie potrzebowała więcej iteracji do nauki. Dla moich testów sieć najszybciej się uczyła dla współczynnika bezwładności 0.9.

**Wnioski:**

Analiza pokazała, że wyższy współczynnik bezwładności spowodował zwiększenie wartości poprawki wag. Jednak nie można także wybrać zbyt dużego współczynnika bezwładności, gdyż może to zwiększyć liczbę potrzebnych iteracji.

Algorytm wstecznej propagacji błędu wykazuje silną zależność pomiędzy przebiegiem procesu trenowania sieci, a wartością współczynnika uczenia. Zbyt mała wartość współczynnika uczenia znacznie zwiększa czas przebiegu treningu, dodatkowo wykrywa literę z gorszym wynikiem.

W procesie uczenia sieci ważna jest inicjalizacja początkowego układu wag połączeń między elementami sieci. Najskuteczniejszym rozwiązaniem jest losowy dobór wag, co praktycznie całkowicie zabezpiecza przed problemem z symetrią.