Dokumentacja implementacji algorytmu wstecznej propagacji na przykładzie funkcji NXoR

Michał Korniak

Spis treści

1 Wprowadzenie

Algorytm wstecznej propagacji błędu jest metodą uczenia wielowarstwowych polegającą na korekcie wag połączeń między neuronami na podstawie błędu całej sieci. W tym dokumencie prześledzę działanie tego algorytmu na przykładzie nauczanie działania funkcji NXoR. jednocześnie przedstawiając jego autorską implementację w języku C#.

2 Architektura sieci dla funkcji NXoR

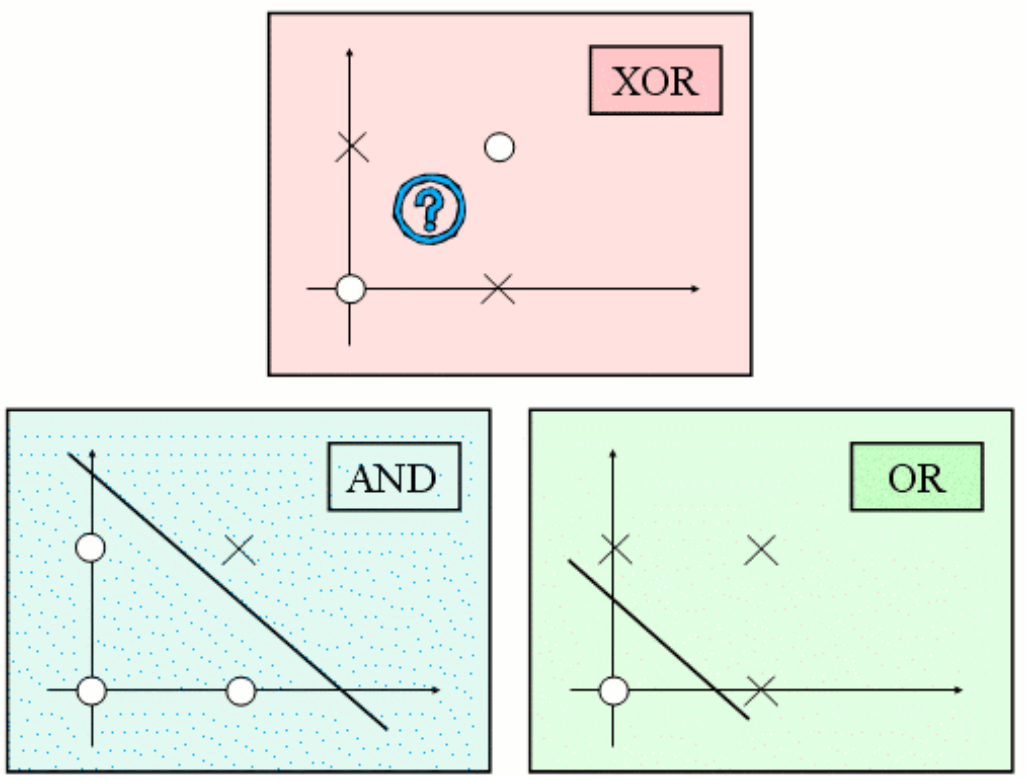
Algorytm wstecznej propagacji jest wykorzystywany do nauczania wielowarstwowych sieci jednokierunkowych. Taka sieć zawiera warstwę wejściową i warstwę wyjściową, może posiadać również warstwy ukryte (wykorzystywana do problemów liniowo nieseparowalnych). Każda warstwa zawiera dowolną ilość neuronów. Neurony są połączone ze sobą w taki sposób, że każdy neuron warstwy innej niż wyjściowa jest połączony z każdym neuronem kolejnej warstwy, a każde połączenie ma określoną wagę. Dodatkowo możliwe jest połączenie do tego zwanego biasa, czyli połączenia do neuronu, który zawsze przyjmuje wartość 1. Połączenia wejściowe do neuronu będą wpływać na to jaką będzie miał wartość.

Kluczowym pytaniem jest to jak powinna wyglądać sieć obsługująca funkcję NXoR. Jako, że funkcja NXoR przyjmuje dwie liczby wejściu jasne jest, że powinna posiadać również dwa neurony wejściowe. Wiemy również, że sieć posiada jedno wyjście, co sprawia, że potrzebujemy tylko jednego neuronu wyjściowego.

x1	x2	NXoR
0	0	1
0	1	0
1	0	0
1	1	1

W powyższym opisie pojawiła się wzmianka o tym, że warstwa ukryta jest wykorzystywana w problemach, które nie są liniowo separowalne. W związku z tym należy się zastanowić czy problem NXoR do takowych należy. Aby sprawdzić czy problem jest liniowo separowalny należy narysować na płaszczyźnie punkty w których funkcja przyjmuje 0 i tych w których przyjmuje 1, a następnie sprawdzić czy istnieje prosta, która przedzieli te punkty

na dwie płaszczyzny. Jak pokazuje rysunek 1 do funkcji spełniających ten warunek należą funkcje AND i OR. Natomiast przykładem funkcji niespełniającej tego warunku jest XOR, a więc też jego odwrotność czyli NXoR. W związku z tym będziemy potrzebować warstwy ukrytej.



Źródło: https://edux.pjwstk.edu.pl/mat/2144/lec/rW2.htm/

Rysunek 1: Liniowa separowalność funkcji AND, OR, XOR

Otwartym pytaniem pozostaje natomiast to ile neuronów powinna posiadać ta warstwa. Tutaj posłużymy się cytatem z książki Jeffa Heatona "Introduction to Neural Networks for C#":

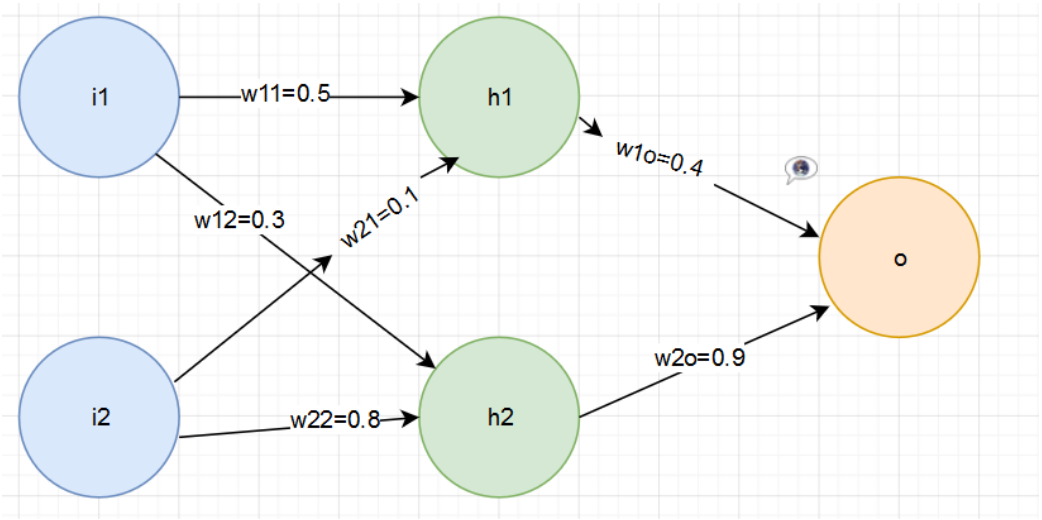
> There are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers, such as the following:
>
> - The number of hidden neurons should be between the size of the input layer and the size of the output layer.

- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

W związku z powyższym nasza warstwa ukryta będzie zawierała 2 neurony.

Kolejnym pytaniem jest to czy należy wprowadzać biasa. Wstępne testy pokazały, że jego obecność przyśpiesza trening sieci, ale aby nie komplikować opisu zostanie on pominięty.

Ostatecznie nasza sieć będzie wyglądała tak jak na rysunku 1



Rysunek 2: Architektura sieci realizującej funkcję NXoR

3 Obliczanie wyjść sieci neuronowej

Nieodzownym krokiem treningu sieci neuronowej jest obliczenie aktualnych wyników i porównanie ich z oczekiwanymi. Przedstawmy to na przykładzie wejścia (0,1), którego oczekiwanym wyjściem jest 0. Informację o wejściu podajemy neuronom wejściowym, tak więc:$i1 = 0$, $i2 = 1$.

Aby obliczyć wartości kolejnych neuronów musimy zsumować wartości połączeń do tego neuronu, a następnie poddając wynik funkcji aktywacji. Przez

wartość połączenia będziemy rozumieli iloczyn wagi i neuronu wejściowego dla połączenia. Wagi połączeń zostały przedstawione na rysunku 1.

Przykładowo dla neuronu h1 wartość przed użyciem funkcji aktywacji będzie równa:

$$net_{h1} = i1 * w11 + i2 * w21$$

Co po podstawieniu wartości da:

$$net_{h1} = 0 * 0.5 + 1 * 0.1 = 0.1$$

Taka wartość jest następnie poddawana działaniu funkcji aktywacji, co umożliwia normalizowanie wartości neuronów do oczekiwanych przedziałów wartości. W tym przypadku skorzystamy z funkcji sigmoidalnej, której zakres wartości mieści się w przedziale [0,1]:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Po podstawieniu otrzymanej wcześniej wartości do funkcji dostaniemy wartość neuronu h1:

$$out_{h1} = f(net_{h1}) = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.1}} \approx 0.525$$

W ten sposób będziemy liczyć wartość kolejnych neuronów:

$$out_{h2} = f(i1 * w12 + i2 * w22) = f(0 * 0.3 + 1 * 0.8) = f(0.8) \approx 0.690$$

$$out_o = f(h1 * w1o + h2 * w2o) = f(0.525 * 0.4 + 0.690 * 0.9) = f(0.831) \approx 0.697$$

4 Trening sieci neuronowej

Jak widzimy otrzymany wynik różni się od oczekiwanego. Dlatego też przeprowadzimy trening sieci, którego celem będzie ustawienie wag w ten sposób, żeby wyjście sieci jak najbardziej odpowiadało wartości oczekiwanej.

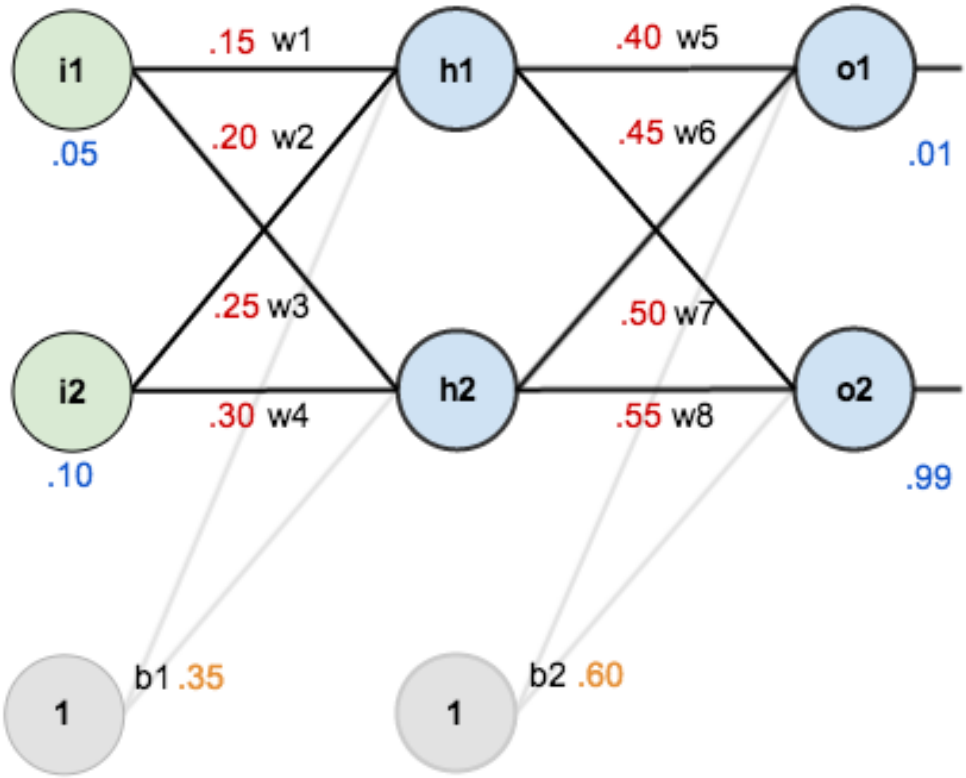
Mając obliczone wyjście sieci oraz wartością oczekiwaną możemy obliczyć wartość błędu. Robimy to przy pomocy funkcji błędu średniokwadratowego.

$$E(O) = 1/n * (d - y)^2$$

gdzie:

- E(O) to błąd średniokwadratowy dla konkretnego neuronu wyjściowego
- n to liczba neuronów warstwy wyjściowej
- d to oczekiwane wyjście neuronu
- y to rzeczywiste wyjście neuronu

Suma błędów średniokwadratowych wszystkich neuronów wyjściowych jest błędem całkowitym.



Źródło: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Rysunek 3: Przykład wielowarstwowej sieci jednokierunkowej z podanymi wejściami oraz oczekiwanymi wyjściami

Rozważmy sieć z rysunku 2, na którym zostały podane wejścia oraz oczekiwane wyjścia sieci. Wartości neuronów warstwy wyjściowej będą równe:

$$out_{o1} \approx 0.7513$$

$$out_{o2} \approx 0.7729$$

Mając podane oczekiwane oraz rzeczywiste wyjście możemy obliczyć błąd średniokwadratowy dla obu neuronów:

$$E_{o1} = 1/2 * (0.01 - 0.7513)^2 \approx 0.274762845$$

$$E_{o2} = 1/2 * (0.99 - 0.7729)^2 \approx 0.023566205$$

Tym samym błąd całkowity wynosi:

$$E_{total} = E_{o1} + E_{o2} \approx 0.274762845 + 0.023566205 \approx 0.29832905$$

Jak widzimy sieć jest daleka do ideału, dlatego będziemy chcieli zmodyfikować wagi tak, aby jak najbardziej zminimalizować błąd. Robimy to za pomocą wzoru:

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

gdzie:

w_5^+ jest nową wartością wagi.

w_5 jest aktualną wartością wagi.

η jest stałą uczenia, czyli wartością, która określa szybkość uczenia sieci.

$\frac{\partial E_{total}}{\partial w_5}$ jest pochodną częściową błędu całkowitego sieci do danej wagi.

Szczególną uwagę należy poświecić stałej uczenia η. Jest ona ustawiana w momencie rozpoczynania procesu, przyjmując wartości z zakresu [0,1]. W zależności od wybranej wartości, zmiany wag będą postępować gwałtowanie lub też łagodnie. Zazwyczaj stała uczenia jest ustawiane na wartości mniejsze od 0.1. Jednakże w tym przykładzie ustawimy ją na 0.5.

Kolejną rzeczą wartą uwagi jest pochodna błędu całkowitego do konkretnej wagi. Liczymy ją z wykorzystaniem reguły łańcuchowej, w przypadku w_5:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Już teraz możemy zauważyć, że wzór na pochodną cząstkową dla w6 będzie bardzo podobny:

$$\frac{\partial E_{total}}{\partial w_6} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_6}$$

Wynika to z tego, że oba połączenie mają taki neuron docelowy. Aby nie liczyć tego samego dwa razy, możemy wprowadzić oznaczenie dla części wspólnej równania:

$$\delta_{o1} = \frac{\partial E_{total}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}}$$

tym samym:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_6} = \delta_{o1} * \frac{\partial net_{o1}}{\partial w_6}$$

Spróbujmy teraz uprościć wyrażenie $\frac{\partial net_{o1}}{\partial w_5}$:

$$\frac{\partial net_{o1}}{\partial w_5} = \frac{\partial(w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1)}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1}$$

Jak widzimy $\frac{\partial net_{o1}}{\partial w_5}$ ostatecznie sprowadza się do wartości neuronu źródłowego dla połączenia. W analogiczny sposób upraszcza się równanie dla $\frac{\partial net_{o1}}{\partial w_6}$, dzięki czemu otrzymujemy:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} * out_{h1}$$

$$\frac{\partial E_{total}}{\partial w_6} = \delta_{o1} * out_{h2}$$

Ostatnim krokiem potrzebnym do wyliczenia nowych wag w_5 jest obliczenie samego współczynnika delty dla o_1 (δ_{o1}):

$$\delta_{o1} = \frac{\partial E_{total}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}}$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial(\frac{1}{2}(d_{o1} - out_{o1})^2 + \frac{1}{2}(d_{o2} - out_{o2})^2)}{\partial out_{o1}} \tag{1}$$

$$= 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0 \tag{2}$$

$$= -(target_{o1} - out_{o1}) \tag{3}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial(\frac{1}{1+e^{-net_{o1}}})}{\partial net_{o1}} \tag{1}$$

$$= (\frac{1}{1 + e^{-net_{o1}}})(1 - \frac{1}{1 + e^{-net_{o1}}}) \tag{2}$$

gdzie:

$$\frac{1}{1 + e^{-net_{o1}}} = out_o1 \tag{3}$$

a więc:

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) \tag{4}$$

W związku z powyższymi równaniami wzór na współczynnik delty dla out_o1 wynosi:

$$\delta_{o1} = \frac{\partial E_{total}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} \tag{1}$$

$$= -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) \tag{2}$$

$$\tag{3}$$

$$\delta_{o1} = \frac{\partial E_{total}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} \tag{1}$$

$$= -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) \tag{2}$$

$$\tag{3}$$

Co podstawieniu daje:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} * out_{h1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) \approx 0.1385$$

W ten sposób otrzymaliśmy łatwy wzór na obliczenie pochodnych cząstkowej po w5 i w6

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} * out_{h1} \approx 0.1385 * 0.593269992 \approx 0.082167$$

$$\frac{\partial E_{total}}{\partial w_6} = \delta_{o1} * out_{h2} \approx 0.1385 * 0.596884378 \approx 0.082668$$

co umożliwia nam obliczenie nowych wag:

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167 = 0.359$$

$$w_6^+ = w_6 - \eta * \frac{\partial E_{total}}{\partial w_6} = 0.45 - 0.5 * 0.082668 = 0.409$$

W analogiczny sposób liczymy nowe wagi połączeń z neuronem o_2:

$$w_7^+ = w_7 - \eta * \frac{\partial E_{total}}{\partial w_7} = w_7 - \eta * \delta_{o2} * out_{h1} = 0.511$$

$$w_8^+ = w_8 - \eta * \frac{\partial E_{total}}{\partial w_8} = w_8 - \eta * \delta_{o2} * out_{h2} = 0.561$$

Bias jest trochę innych w przypadkiem, gdyż jest połączony jednocześnie z neuronami o1 i o2, w związku z tym pochodna cząstkowa do tej wagi będzie wyliczana w następujący sposób:

$$\frac{\partial E_{total}}{\partial b_2} = \frac{\partial E_1 + E_2}{\partial b_2} = \frac{\partial E_1}{\partial b_2} + \frac{\partial E_2}{\partial b_2} \tag{1}$$

$$= (\frac{\partial E_1}{\partial b_2}) \tag{2}$$

$$\tag{3}$$

$$b_2^+ = b_2 - \eta * \frac{\partial E_{total}}{\partial b_2} = b_2 - \eta * \delta_{o1} + \delta_{o_2} * out_{h2} = 0.561370121$$