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 sieci realizującej funkcję NXoR, jednocześnie przedstawiając jego autorską implementację w języku C#.

2 Algorytm wstecznej propagacji

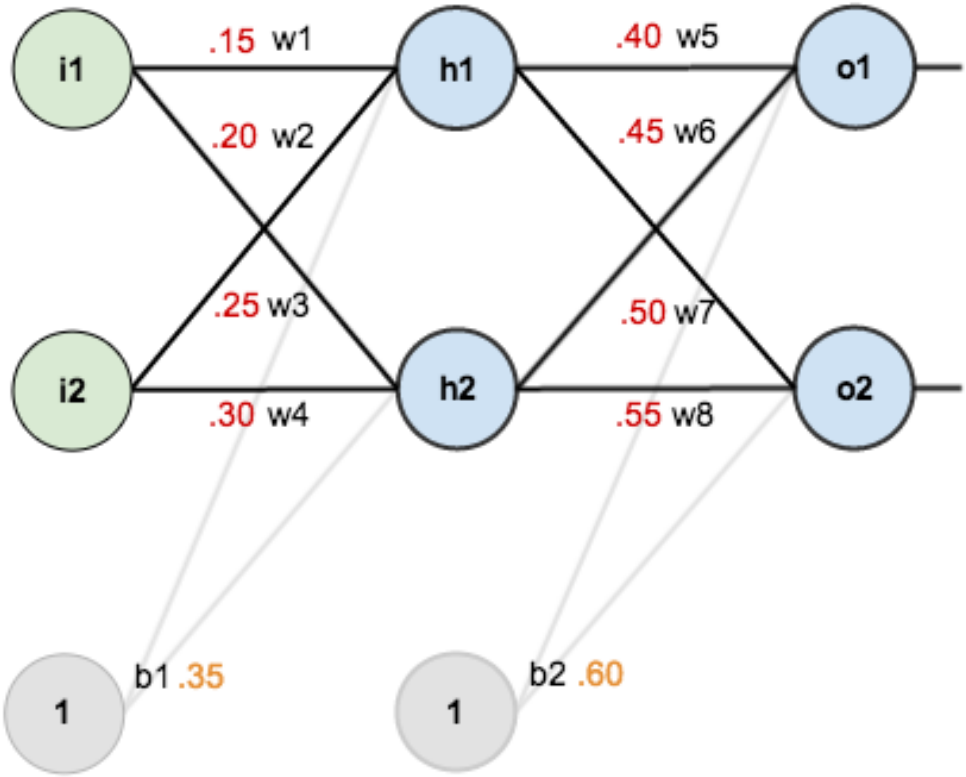
2.1 Sposób działania wielowarstwowych sieci neuronowych

Algorytm wstecznej propagacji jest wykorzystywany do nauczania wielowarstwowych sieci jednokierunkowych. Taka sieć zawiera warstwę wejściową i warstwę wyjściową, może posiadać również warstwy ukryte (w zależności od problemu jedną albo dwie). 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.

Rysunek 1 przedstawia przykład jednokierunkowej sieci neuronowej opartej o dwa neurony warstwy wejściowej i tyle samo neuronów w warstwach ukrytej i wyjściowej. Oprócz połączeń do innych neuronów, istnieją również połączenia do biasa, który jest wspólny dla neuronów w ramach jednej warstwy.

Rysunek 1 zawiera wartości neuronów wejściowych oraz wag, dzięki czemu będziemy mogli pokazać jak wyliczane są wartości kolejnych neuronów. Robimy to sumując wartości połączeń do tego wektora, a następnie poddając wynik funkcji aktywacji. Przez wartość połączenia będziemy rozumieli iloczyn wagi i neuronu wejściowego dla połączenia. Przykładowo dla neuronu h1 wartość przed użyciem funkcji aktywacji będzie równa:

$$net = i1 * w1 + i2 * w2 + 1 * b1$$



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

Rysunek 1: Przykład wielowarstwowej sieci jednokierunkowej

Co po podstawieniu wartości da:

$$net_{h1} = 0.05 * 0.15 + 0.10 * 0.20 + 1 * 0.35 = 0.3775$$

Taka wartość jest następnie poddawana działaniu funkcji aktywacji. Takie działanie 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:

$$out_{h1} = f(net_{h1}) = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.3775}} \approx 0.5932$$

W ten sposób będziemy liczyć wartość kolejnych neuronów aż otrzymamy wynik.

2.2 Implementacja wielowarstwowej sieci neuronowej

Teraz przedstawię implementację mechanizmu zaprezentowanego w poprzednim podrozdziale. Jej podstawą jest klasa odwzorowującej sieć neuronową, w naszym przypadku taką funkcję pełni klasa NeuralNetwork, która to implementuje interfejs pokazany na listingu 1.

Listing 1: Interfejs INeuralNetwork

```
1  public interface INeuralNetwork
2  {
3      IEnumerable<IInputNeuron> InputLayer { get; }
4      IEnumerable<IHiddenNeuron> HiddenLayer { get; }
5      IEnumerable<IOutputNeuron> OutputLayer { get; }
6      IErrorFunction ErrorFunction { get; }
7      void FillInputNeurons(IEnumerable<double> input);
8      IEnumerable<double> CalculateOutput();
9  }
```

W tym momencie skupmy się na linijkach 3, 4 i 5, które to wskazują na to, że sieć neuronowa zawiera warstwy wejściową, ukrytą i wyjściową. Jak widzimy każda warstwa posiada inny typ neuronu, co wynika to z tego, że w zależności od warstwy neurony się różnią. Przykładowo:

- Neuron warstwy wejściowej nie ma możliwości dodawanie połączeń wejściowych.
- Neuron warstwy wyjściowej nie ma możliwości dodawanie połączeń wyjściowych.
- Neuron warstwy ukrytej ma możliwość dodawanie obu typów połączeń.

Obiekt klasy NeuralNetwork jest tworzony z wykorzystaniem wzorca "Builder". Dzieje się to w sposób pokazany na listingu 2.

Listing 2: Budowanie obiektu NeuralNetwork

```
var neuralNetworkBuilder = new NeuralNetworkBuilder();
var network = neuralNetworkBuilder
    .SetNumberOfInputNeurons(2)
    .SetNumberOfOutputNeurons(1)
    .SetActivationFunction(new SigmoidActivationFunction())
    .SetErrorFunction(new MeanSquaredErrorFunction(1))
    .SetNumberOfHiddenNeurons(3)   //opcjonalne
    .AddBiasConnections()          //opcjonalne
    .Build();
```

Klasa NeuralNetworkBuilder umożliwia stworzenie sieci neuronowej:

- Zawierającą wybraną ilość neuronów warstwy wejściowej.
- Zawierającą wybraną ilość neuronów warstwy wyjściowej
- Działającą na określonej funkcji aktywacji
- Wyliczającej błąd na podstawie wybranej funkcji
- Umożliwiającej opcjonalne dodanie warstwy ukrytej
- Umożliwiającej opcjonalne dodanie połączeń do biasów

Zadaniem Buildera jest stworzenie obiektu klasy NeuralNetwork, który spełni podane wymagania. W zależności od tego czy użytkownik będzie potrzebował warstwy ukrytej, Builder utworzy sieć która ma połączenia z warstwą pośrednią lub też bezpośrednie połączenia warstwy wejściowej z wyjściowej. Podobnie jest z wyborem tego czy czy powinny być tworzone biasy czy też nie. Builder nie posiada żadnej metody dotyczącej wyboru pierwotnych wag połączeń, wynika to z tego, że zgodnie z założeniem algorytmu wstecznej propagacji są one inicjalizowane losowo.

Dla tak stworzonej sieci neuronowej możemy przeprowadzić obliczenia, które zostały przedstawione w poprzednim podrozdziale. Robimy to w sposób przedstawiony w listingu 3

Listing 3: Budowanie obiektu NeuralNetwork

```
network.FillInputNeurons(new double[] { 0, 1 });
var output = network.CalculateOutput();
```

W pierwszej kolejności są wypełniane neurony wejściowe. Jeśli użytkownik poda liczbę danych wejściowych różniącą się od liczby neuronów warstwy wejściowej zostanie wyrzucony wyjątek.

Wyjście sieci jest liczone w sposób rekurencyjny:

- Wyjście każdego neuronu z wyjątkiem neuronów wejściowych jest sumą wyjść połączeń wejściowych poddaną funkcji aktywacji (listing 4).
- Wyjście połączenia jest iloczynem wagi i wyjścia neuronu źródłowego (listing 5)

Listing 4: Budowanie obiektu NeuralNetwork

```
1  public class OutputNeuron : IOutputNeuron
2  {
3      //***
4      public double NetOutput => _inputConnections.Sum(x => x.Output);
5      public double Output => _activationFunction.Invoke(NetOutput);
6      //***
7  }
```

Listing 5: Budowanie obiektu NeuralNetwork

```
1  class NeuronConnection: INeuronConnection
2  {
3      //***
4      public double Output => Weight * _source.Output;
5      //***
6  }
```

2.3 Trening sieci neuronowej

Wagi połączeń są inicjowane wartościami losowymi, w związku z tym możemy się spodziewać, że pierwotne wyjście sieci będzie znacznie odbiegało od tego czego oczekujemy. Dlatego też przeprowadzany jest trening sieci, którego celem jest ustawienie wag w ten sposób, żeby wyjście sieci jak najbardziej odpowiadało wartościom oczekiwanym.

Pierwszym krokiem jest podanie sieci danych testowych, które posiadają informację na temat wejścia oraz oczekiwanego wyjścia sieci. Następnie obliczane jest rzeczywiste wyjście neuronu, które jest porównywane z oczekiwanym za pomocą funkcji błędu średniokwadratowego.

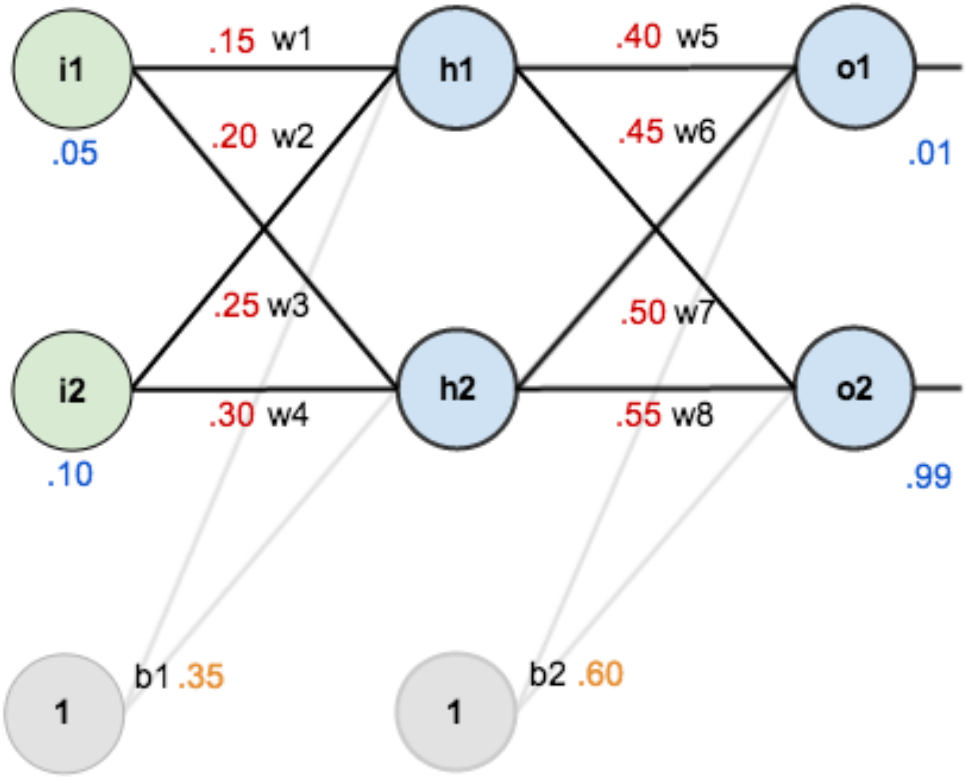
$$E(O) = 1/n * (d - y)^2$$

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

Sumę błędów średniokwadratowych wszystkich neuronów wyjściowych jest błędem całkowitym.



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

Rysunek 2: 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 zminimalizować błąd całkowity.