**Scenariusz 1**

Robert Białas, Gr. 1

**Temat ćwiczenia:  
Budowa i działanie Perceptronu**

**1. Cel Ćwiczenia**

Celem ćwiczenia było poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

**2. Opis budowy perceptronu oraz wykorzystanego algorytmu**

Perceptron został zaimplementowany wg modelu McCullocha-Pittsa omówionego w książce „Sieci Neuronowe do przetwarzania informacji” Stanisława Osowskiego.

Sygnały wejściowe xi (i = 1, 2, …, N) zawarte w tablicach ONE\_ONE, ONE\_ZERO, ZERO\_ONE, ZERO\_ZERO są sumowane z odpowiednimi wagami wi  w sumatorze, tj. funkcji Perceptron::getResult(). Sygnał wyjściowy neuronu yj można opisać wzorem:

gdzie funkcje , czyli funkcję aktywacyjną wyrażamy wzorem:



|  |
| --- |
| Main.cpp:  double  ONE\_ONE[] ={bias,1,1},  ZERO\_ONE[] ={bias,0,1},  ONE\_ZERO[] ={bias,1,0},  ZERO\_ZERO[] ={bias,0,0}; |
| Perceptron.cpp:  int Perceptron::getResult(const double inputs[])  {  double sum=0;  for(int i=0; i<this->numberOfInputs; i++)  sum+=this->weights[i]\*inputs[i];//suma wejść przemnożonych przez wagi  //if(sum>1) return 1;//FUNKCJA AKTYWACYJNA  //else return 0;  return(sum>1) ? 1:0;  } |

Model ten jest modelem matematycznym, w którym stan neuronu określony jest na podstawie stanu sygnałów wejściowych neuronów w chwili poprzedniej.

Zaimplementowany perceptron jest w stanie nauczyć się wybranej funkcji logicznej AND, NAND, OR lub NOR. Można również podjąć próbę nauki funkcji XOR, ale dla perceptronu jednowarstwowego nie jest możliwe uzyskanie poprawnych wyników gdyż XOR nie jest linearnie podzielny.

|  |
| --- |
| Main.cpp:  int \*function = new int[3];  string functionNames[] ={"AND","NAND","OR","XOR","NOR"};  cin>>functionIndex;  LogicFunction(functionIndex,function);  Perceptron perceptron(numberOfInputs,learningRate);  cout << functionNames[functionIndex-1] << " - wyniki PRZED uczeniem" << endl;  cout << "Wejscie (0,0) - wynik: " << perceptron.getResult(ZERO\_ZERO) << endl;  cout << "Wejscie (0,1) - wynik: " << perceptron.getResult(ZERO\_ONE) << endl;  cout << "Wejscie (1,0) - wynik: " << perceptron.getResult(ONE\_ZERO) << endl;  cout << "Wejscie (1,1) - wynik: " << perceptron.getResult(ONE\_ONE) << endl;  //for(int i=0; i<numberOfEpochs; i++)  bool done=false;  while(done==false)  {  numberOfEpochs++;  done=true;  perceptron.learn(ZERO\_ZERO,function[0]);  perceptron.learn(ZERO\_ONE,function[1]);  perceptron.learn(ONE\_ZERO,function[2]);  perceptron.learn(ONE\_ONE,function[3]);  if(perceptron.getResult(ZERO\_ZERO)!=function[0])done=false;  if(perceptron.getResult(ZERO\_ONE)!=function[1])done=false;  if(perceptron.getResult(ONE\_ZERO)!=function[2])done=false;  if(perceptron.getResult(ONE\_ONE)!=function[3])done=false;  }  cout << endl << functionNames[functionIndex-1] << " - wyniki PO uczeniu" << endl;  cout << "Wejscie (0,0) - wynik: " << perceptron.getResult(ZERO\_ZERO) << endl;  cout << "Wejscie (0,1) - wynik: " << perceptron.getResult(ZERO\_ONE) << endl;  cout << "Wejscie (1,0) - wynik: " << perceptron.getResult(ONE\_ZERO) << endl;  cout << "Wejscie (1,1) - wynik: " << perceptron.getResult(ONE\_ONE) << endl;  cout << "Liczba wykonanych epok: " << numberOfEpochs << endl;  -------------------------------------------------------  void LogicFunction(int &functionIndex,int function[])  {  switch(functionIndex)  {  case 1://AND  function[0]=0; function[1]=0; function[2]=0; function[3]=1; break;  case 2://NAND  function[0]=1; function[1]=1; function[2]=1; function[3]=0; break;  case 3://OR  function[0]=0; function[1]=1; function[2]=1; function[3]=1; break;  case 4://XOR  function[0]=0; function[1]=1; function[2]=1; function[3]=0; break;  case 5://NOR  function[0]=1; function[1]=0; function[2]=0; function[3]=0; break;  default://DOMY?LNA FUNKCJA TO AND  function[0]=0; function[1]=0; function[2]=0; function[3]=1; functionIndex=1; break;  }  } |

Wagi stanowią losowe liczby z zakresu (0; 1) i są wyliczane przy pomocy funkcji Perceptron::getRandomDouble(). Dzięki nim obliczamy wartość sygnału wyjściowego yi.

|  |
| --- |
| Perceptron.cpp:  double Perceptron::getRandomDouble()  {  return ((double)rand()/(double)RAND\_MAX);  }  Perceptron::Perceptron() {//Konstruktor domyślny  srand(time(NULL));  this->learningRate = 0.01;  this->numberOfInputs = 3;  if(this->numberOfInputs <= 0)  this->numberOfInputs = 1;  this->weights = new double[numberOfInputs];//tworzymy tablice wag, zalezna od ilosci wejsc  for(int i = 0; i < this->numberOfInputs; i++)  this->weights[i] = getRandomDouble();//Wagi wi są przypadkowymi liczbami zmiennoprzecinkowymi  } |

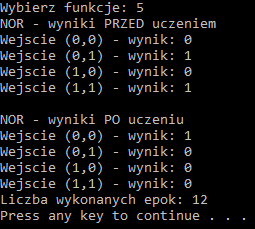
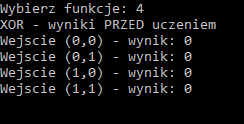
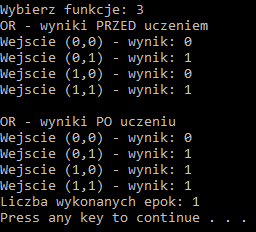
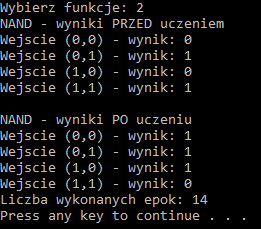
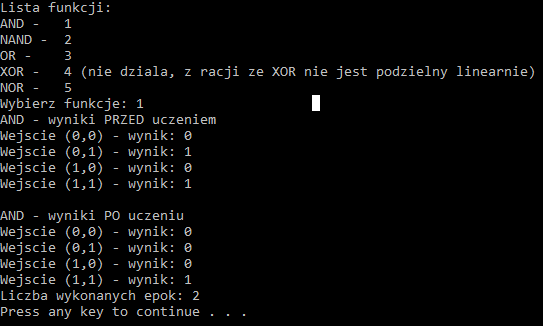
W funkcji uczenia, czyli Perceptron::learn(), dokonywane jest sprawdzenie czy aktualna wartość yi jest równa wartości oczekiwanej di . Wartość aktualna jest ustalana za pomocą funkcji Perceptron::getResult(), w której znajduje się sumator oraz funkcja aktywacyjna. Jeśli wartość aktualna i oczekiwana nie są równe, to wtedy wagi zostają zaktualizowane w funkcji Perceptron::changeWeights() poprzez przemnożenie współczynnika uczenia przez wejścia xi oraz błąd lokalny di – yi . Podczas uczenia wykorzystuje się jedynie informacje o aktualnej wartości sygnału wyjściowego neuronu oraz z wag.

|  |
| --- |
| //ucz sie, czyli:  void Perceptron::learn(const double inputs[],int expectedResult) {  int result = getResult(inputs);//otrzymaj wyniki, czyli zsumuj kazde wi\*xi i przekaz sumę funkcji aktywacyjnej  if(result == expectedResult)//Jesli wynik sie zgadza z pozadanym...  return;//Zatrzymujemy uczenie  changeWeights(result,expectedResult,inputs);//W przeciwnym wypadku zmieniamy wagi  }  int Perceptron::getResult(const double inputs[])  {  double sum=0;  for(int i=0; i<this->numberOfInputs; i++)  sum+=this->weights[i]\*inputs[i];//suma wejść przemnożonych przez wagi  //if(sum>1) return 1;//FUNKCJA AKTYWACYJNA  //else return 0;  return(sum>1) ? 1:0;  }  void Perceptron::changeWeights(int actualResult,int desiredResult,const double inputs[])  {  for(int i=0; i<this->numberOfInputs; i++)  this->weights[i]+=this->learningRate\*(desiredResult-actualResult)\*inputs[i];//współczynnik uczenia\*błąd lokalny\*xi  }//deltawij = xj(di-yi) <- Osowski, (2.4) str. 19; Działanie uczenia jest przyśpieszone poprzez pomnożenie deltawij przez współczynnik uczenia |

Perceptron jest inicjalizowany poprzez podanie do konstruktora ilości danych uczących oraz wskaźnika uczenia, następnie w pętli, uczy się do momentu aż wartości oczekiwane pokrywają się z wartościami oczekiwanymi.

|  |
| --- |
| Perceptron::Perceptron(unsigned numOfInputs,double trainingRate) {  srand(time(NULL));  this->learningRate = trainingRate;  this->numberOfInputs = numOfInputs;  if(this->numberOfInputs <= 0)  this->numberOfInputs = 1;  this->weights = new double[numberOfInputs];//tworzymy tablice wag, zalezna od ilosci wejsc  for(int i = 0; i < this->numberOfInputs; i++)  this->weights[i] = getRandomDouble();//Wagi wi są przypadkowymi liczbami zmiennoprzecinkowymi  } |

**3. Otrzymane wyniki dla współczynnika uczenia równego 0.1 oraz ilości danych równych 3:**



**4. Zestawienie wyników:**

**Tabela ilości epok potrzebnych do nauczenia perceptronu w zależności od funkcji:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Funkcja | Próba 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Średnia |
| AND | 15 | 29 | 1 | 9 | 22 | 10 | 15 | 12 | 16 | 13 | 14 |
| NAND | 113 | 125 | 103 | 88 | 123 | 104 | 91 | 128 | 100 | 140 | 112 |
| OR | 11 | 22 | 22 | 13 | 5 | 13 | 21 | 21 | 15 | 33 | 18 |
| NOR | 138 | 104 | 151 | 186 | 183 | 181 | 178 | 106 | 182 | 100 | 151 |

**5. Wnioski:**

* Perceptron stanowi abstrakcyjny model sieci neuronowej.
* Perceptron ma swoje ograniczenia – w pojedynczej warstwie nie może się nauczyć funkcji które nie są linearnie separowalne (np. XOR)
* Ilość wykonanych epok jest wysoce zależna od ustawionych wag – losowe wagi powodują kompletnie losowe wyniki, uniemożliwiające ustalenie zależności/wykresów w kontekście, pojedynczej, określonej funkcji.
* Każda funkcja ma inne wymagania co do ilości epok koniecznych do nauczenia:

1. **NOR (Średnio 151 epok)**
2. **OR (112)**
3. **NAND (18)**
4. **AND (14)**