Sieci Neuronowe

Zadanie 1

Narysuj wykresy następujących funkcji aktywacji:

A. Funkcji sigmoidalnej
B. Funkcji tangensa hiperbolicznego
C. Funkcji ReLU

Mając dany perceptron i dowolną z tych funkcji aktywacji powiedz jaka jest rola wyrazu wolnego, który jest dodawany do wejść sieci neuronowej. Jakie byłyby konsekwencje jego braku?

Zadanie 2

Napisz wywołanie funkcji `perceptron`, która będzie realizowała obliczenia perceptronu z dwoma wejściami x_1 oraz x_2, z progową funkcją aktywacji. Jako wejście będzie przyjmowała wagi w_1, w_2 oraz wejścia $x_{1,}x_2 \in \{0,1\}$ Dobierz w sposób manualny wagi w1, w2 oraz w3 do problemu

```
> perceptron <- function(w1,w2,w3,x1,x2){
  y <- w1*x1+w2*x2 + w3
  if(y>0){
    1
  } else {
    0
  }
}
```

1. Operacji logicznej AND:

   Przykład: parametryzacja postaci w1=0.2, w2=0,w3=0.1 nie jest dobra mimo, że dla x1=1 oraz dla x2=1 wartość $x_1 AND\, x_2$ jest poprawna to już dla x1=0 oraz x2=1 wartość dalej wynosi 1, podczas gdy powinna 0.

x_1	x_2	$x_1 AND\, x_2$
0	0	0
0	1	0
1	0	0
1	1	1

2. Operacji logicznej OR:

x_1	x_2	$x_1 OR\, x_2$
0	0	0
0	1	1
1	0	1
1	1	1

3. Czy podobnie uda się dobrać wagi dla problemu XOR? Odpowiedź uzasadnij.

x_1	x_2	$x_1 XOR\, x_2$
0	0	0
0	1	1
1	0	1
1	1	0

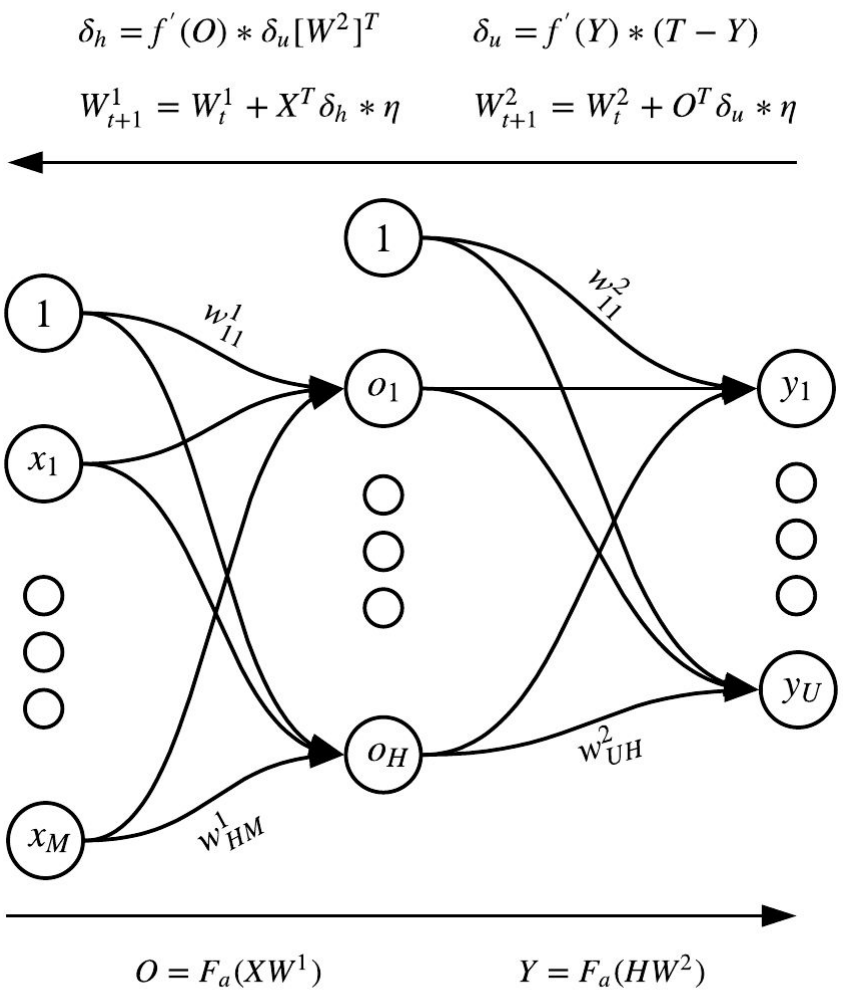
W kolejnych zadaniach Twoim będziemy zajmować się konstruowaniem 3-warstwowej sieci neuronowej od dowolnej liczbie neuronów w warstwie wejściowej, ukrytej oraz wyjściowej. Sieć będziemy uczyć w trybie batchowym. Załóżmy, że przykładowe macierze x (zmienne objaśniające, zbiór treningowy) oraz y (zmienne objaśniane, zbiór treningowy) mają w wierszach przykłady a w kolumnach cechy. Dla przykładu, problem koniunkcji logicznej w postaci macierzy x,y będzie reprezentowany następująco:

```
> x
     [,1] [,2]
[1,]    0    0
[2,]    0    1
[3,]    1    0
[4,]    1    1
> y
     [,1]
[1,]    0
[2,]    0
[3,]    0
[4,]    1
```

Zadanie 3

W pierwszym kroku napisz procedurę `computeFeedForward`, która na wyjściu zwróci listę zawierającą dwie wartości: ou oraz oh, które odpowiednio oznaczają wyjście z warstwy wyjściowej oraz wyjście z warstwy ukrytej. Funkcja sigmoid znajduje się w pakiecie `e1071`

```
> computeFeedForward <- function(x, w1, w2, actFunc = sigmoid){
  ...
  list(ou = ..., oh=...)
}
```



Zadanie 4

Napisz procedurę `trainANN`, która na wyjściu zwróci listę zawierającą dwie wartości: w1 oraz w2, które odpowiednio oznaczają macierze wag pomiędzy warstwą wejściową a warstwą ukrytą oraz pomiędzy warstwą ukrytą a warstwą wyjściową. Jest to funkcja, której zadaniem jest trenowanie sieci w trybie batchowym. To o czym należy pamiętać to:

1. Zainicjalizowanie zmiennych odpowiedzialnych za przechowywanie macierzy w1 i w2 w sposób losowy.
2. W zmiennej `itNmb` przechowujemy maksymalną liczbę iteracji (jest to brutalny warunek stopu nie odwołujący się do wniosków, które należałoby wyciągać wraz z kolejnymi iteracjami, ale dobry “na początek”)

3. Iterując się po od 1 do `itNmb` należy wykonywać na przemian fazę FeedForward oraz fazę BackPropagation. Pierwsza z nich oblicza wyjście dla zbioru treningowego dla aktualnych wartości macierzy wag w1 oraz w2. Druga aktualizuje te wagi zgodnie z kierunkiem największego spadku.
4. Skorzystaj ze zmiennych pomocniczych aby określić wymiarowość macierzy w1 i w2: `nmbOfInputsInW1, nmbOfInputsInW2, nmbOfOutputs`
5. Implementacja funkcji `computeBackpropagation` będzie przedmiotem kolejnego zdania

```
> trainANN <- function(x, y, hidden = 10, eta = 0.1, itNmb=10000){
  nmbOfInputsInW1 = dim(x)[2]+1
  nmbOfInputsInW2 = (hidden+1)
  nmbOfOutputs = dim(y)[2]
  w1 <- ...
  w2 <- ...
  for(i in 1:itNmb){
    outputs = computeFeedForward(x,w1,w2)
    newW = computeBackpropagation(x,y,outputs$ou,outputs$oh, w1,w2,eta)
    w1 = newW$w1
    w2 = newW$w2
  }
  list(w1=w1, w2=w2, hidden=10)
}
```

Zadanie 5

Zaimplementuj funkcję `computeBackpropagation` realizującą zadanie wstecznej propagacji.

```
> computeBackpropagation <- function(x, y, ou, oh, w1, w2, eta){
  delta_u = ...
  ohB = cbind(oh, rep(1, dim(oh)[1])) #oh + bias

  delta_h = ...
  xB = cbind(x, rep(1,dim(x)[1]))# #x + bias

  w2 <- w2 + ...
  w1 <- w1 + ...

  list(w1 = w1, w2 = w2)
}
```

Zadanie 6

Zaimplementuj funkcję `predict`, której zadaniem jest dla zadanego zbioru przykładów (którego postać macierzowa jest identyczna jak zmiennej x wykorzystywanej do uczenia) obliczyć wartości zmiennej objaśnianej (czyli macierz/wektor postaci macierzowej identycznej jak zmiennej y wykorzystywanej do uczenia).

```
> predict <- function(x, model, actFun = sigmoid){
  x = cbind(x, rep(1,dim(x)[1]))
  h = actFunc( x%*%model$w1 )
  h2 = cbind(h, rep(1,dim(h)[1]))
  out = actFunc(...)
  out
}
```

Zadanie 7

Do dzieła! Teraz skoro mamy już w pełni działającą sieć neuronową możemy jej użyć! Najpierw sprawdźmy czy sieć nauczy się rozwiązywać problem koniunkcji logicznej:

```
> x = matrix(c(0,0,1,1,0,1,0,1), 4)
> y = matrix(c(0,0,0,1))
> model = trainANN(x,y)
> predict(x, model)
```

Zmień postać macierzy x oraz y tak aby nauczyć się problemu XOR z zadania 2.

Zadanie 8

Podobnie jak w zadaniu z SVM załaduj zbiór “cats”:

```
> data(cats, package = "MASS")
```

a) Zbuduj model dla tego zbioru z wykorzystaniem metod do trenowania sieci neuronowych z poprzedniego zadania. Konieczna będzie transformacja danych wejściowych:

```
> x = data.matrix(cats[,-1],rownames.force = NA)
> y = data.matrix(cats[,1], rownames.force = NA)
> y[y[,1] == "F", ] <- 1
```

```
> y[y[,1] == "M", ] <- 0
> y = data.matrix(as.numeric(y), rownames.force = NA)
```

Co zuważyłeś? Czy model, który otrzymałeś wykazuje zdolność uczenia z przedstawionych danych? Do analizy

b) Dokonaj przeskalowania danych zgodnie z poniższą formułą:

```
> x = apply(x, MARGIN = 2, FUN = function(X) (X - min(X))/diff(range(X)))
```

Co zaobserwowałeś? Jeśli skalowanie pomogło - to wyjaśnij dlaczego?

Porównaj wyniki otrzymane z wynikami modelu SVM z biblioteki e1071. Do tego celu użyj funkcji `confusionMatrix` z biblioteki `caret`.