Relatório 3 - Perceptron e Adaline

1 Introdução

Os resultados das funções desse relatório são feitos com base a seguinte base da dados provida pelo professor:

Table 1: Base de Dados

s_1	s_2	t
1.0	1.0	1
1.1	1.5	1
2.5	1.7	-1
1.0	2.0	1
0.3	1.4	1
2.8	1.0	-1
0.8	1.5	1
2.5	0.5	-1
2.3	1.0	-1
0.5	1.1	1
1.9	1.3	-1
2.0	0.9	-1
0.5	1.8	1
2.1	0.6	-1

Em 1960, utilizando o neurônio de McCulloch-Pitts, o psicólogo Frank Rosenblatt contruiu o Mark I Perceptron, esse foi o primeiro computador que aprendia novas habilidades por tentativa e erro.

No mesmo ano, Bernard Widrow e Ted Hoff desenvolveram e implementaram o neurônio Adaline (Adaptive linear neuron). Esse neurônio usa a regra delta ou LMS(least mean square) para o seu treinamento, tendo entradas e saídas contínuas, visando a minimização do erro quadrático calculado pela equação: $E_q = (t\text{-}y)^2$. Para atingir essa minimização é usada a seguinte formula para ajustar os pesos: $w_f\text{-}w_i = a(t\text{-}y)x_i$. Sendo o a a taxa de aprendizado do neurônio. Ademais ele tinha uma função de ativação linear e não degrau como os neurônios que foram implementados anteriormente.

Durante esse relatório foram implementados ambos o perceptron e o adaline.

2 Objetivo

- Treinar um perceptron e um adaline com a base de dados provida pelo professor e obter os pesos e bias finais.
- Plotar os pontos (x1, x2) de treinamento para ambos, e para o adaline plotar o erro quadrático em tempo de treinamento.

3 Desenvolvimento

Para realizar o treinameto de ambas as redes neurais foi utilizada a linguagem Common Lisp. A escolha da linguagem forçou uma abordagem diferente da do exemplo dado, visto que, foi utilizada uma abordagem essencialmente funcional e o exemplo foi feito em uma linguagem procedural. Ademais, foi utilizado a biblioteca vgplot para realizar a plotagem dos gráficos.

Algumas funções criadas foram utilizadas no treinamento de ambas as redes neurais, como por exemplo, a w-separated e a b-separated. Como foi optado por calcular o bias junto com os pesos foi necessária a criação dessas duas funções para que fosse mais prática a separação desses:

```
(defun w-separated (ajusted-weights)
  (butlast ajusted-weights))
(defun b-separated (ajusted-weights)
  (last ajusted-weights))
```

Além disso, outra função que se mostrou necessária no treinamento de ambas as redes foi a yliquid, que como o nome já sugere calcula o valor do y líquido:

```
(defun yliquid (ajusted-weights input)
  (+ (first (b-separated ajusted-weights))
         (apply #'+ (map 'list  #'* (w-separated ajusted-weights) (w-separated input)))))
```

3.1 Perceptron

Para o treinamento do perceptron foram reaproveitadas diversas funções utilizadas para o treinamento do neurônio de Mcculloch-Pitts com a regra de Hebb. Funções como unit-step, ou a funções degrau, foram implementadas da mesma maneira:

```
 (defun unit-step (inputs ajusted-weights threshold)
  (if (>= (yliquid ajusted-weights inputs) threshold) 1 -1))
```

Entretanto, funções como ajust-weights e apply-ajust-weights, que ajustam os pesos e o bias, precisaram de algumas alterações, já que a formula de ajuste de peso havia mudado:

```
(defun ajust-weights (input weights output learning-rate)
  (map 'list #'+ weights
       (map 'list #'(lambda (x) (* (* x output) learning-rate)) input)))

(defun apply-ajust-weights (inputs weights outputs learning-rate threshold)
  (cond
    ((and (null inputs) (null outputs)) weights)
    ((eql
      (first outputs)
      (unit-step (first inputs) weights threshold))
     (apply-ajust-weights (rest inputs) weights (rest outputs) learning-rate threshold))
    (t (apply-ajust-weights
                (rest inputs)
                (ajust-weights (first inputs) weights (first outputs) learning-rate)
                (rest outputs)
                learning-rate
                threshold))))
```

Com todas essas funções prontas, já era possível programar a função perceptron, que é a função de treinamento:

```
(defun perceptron (inputs weights outputs learning-rate threshold )
  (if (equal
           (w-separated weights)
           (w-separated (apply-ajust-weights
                                          inputs weights outputs learning-rate threshold)))
       weights
      (perceptron
       inputs
       (apply-ajust-weights inputs weights outputs learning-rate threshold)
       outputs
       learning-rate
       threshold)))
```

Ao aplicar essa função na base de dados provida pelo professor com pesos e bias iniciais iguais a 0 e alpha igual a 1 foi obtido que os pesos finais são -2.6 e 2.1999998 e o bias final é 1. Ao comparar os resultados obtidos usado esses pesos com os esperados foi comprovado que eles estão certos.

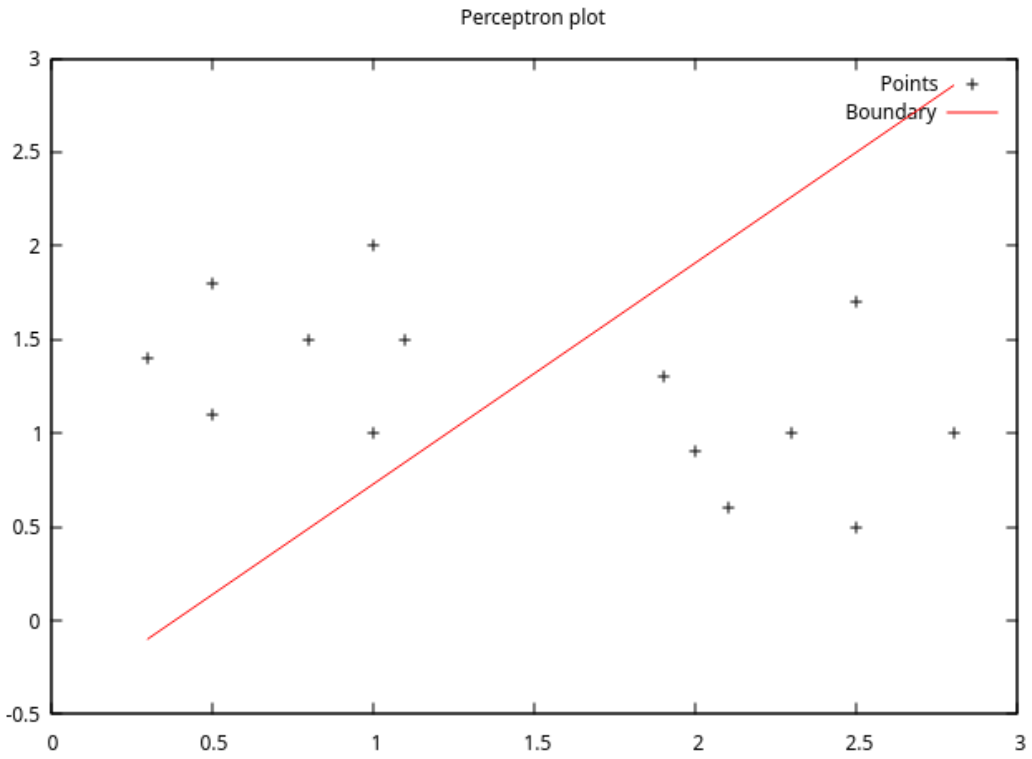
Em posse dos pesos e bias finais foi possível começar a plotagem do gráfico. Para realizar a plotagem da linha de Boundary foi necessária a criação de uma função que realizasse a equação característica dessa reta:

```
(defun equation (weights x1)
  (destructuring-bind  (w1 w2 b) weights
        (/ (- (* (* w1 -1) x1) b) w2)))
```

Com essa equação, foi possível criar as duas funções que realizam a plotagem do gráfico:

```
(defun plot (x y)
  (progn
        (vgplot:plot x y "k+;Points"
                                  '(0.3 2.8)
                                  (list (equation
                                          (perceptron table '(0 0 0) table-output 1 0) 0.3)
                                            (equation
                                          (perceptron table '(0 0 0) table-output 1 0) 2.8))
                                  "r-;Boundary")
        (vgplot:title "Perceptron plot")))

(defun apply-plot (inputs)
  (plot (map 'list #'first inputs) (map 'list #'second inputs)))
```

O resultado da função é o gráfico abaixo, no qual os pontos abaixo da reta produzem saídas negativas e os acima produzem saídas positivas



3.2 Adaline

Para o treinamento do adaline foram reaproveitada as funções ajust-weights e apply-ajust-weights, realizando apenas algumas alterações, visto que a formula de treino do

adaline é diferente da do perceptron:

```
(defun ajust-weights (input weights output learning-rate)
  (map 'list #'+ weights
       (map 'list #'(lambda (x) (* (* x (- output (yliquid weights input))) learning-rate))

(defun apply-ajust-weights (inputs weights outputs learning-rate)
  (cond
        ((and (null inputs) (null outputs)) weights)
        (t (apply-ajust-weights
                (rest inputs)
                (ajust-weights (first inputs) weights (first outputs) learning-rate)
                (rest outputs)
                learning-rate))))
```

Como seria necessário iniciar o adeline com pesos aleatórios foram criadas duas funções para que isso pudesse ser feito de maneira prática:

```
(defun random-weights ()
  (list (make-RANDOM-WEIGHTS) (make-RANDOM-WEIGHTS) 1))

(defun make-random-weights ()
  (- (random 0.5) (random 0.5)))
```

Em posse dessas funções, foi possível criar a função de treinamento do adaline, foi optado por limitar o treinamento com um número máximo de ciclos:

```
(defun adaline(inputs weights outputs learning-rate max-cicles)
  (cond
        ((eql max-cicles 0) weights)
        (t (adaline
                inputs
                (apply-ajust-weights inputs weights outputs learning-rate)
                outputs
                learning-rate
                (- max-cicles 1)))))
```

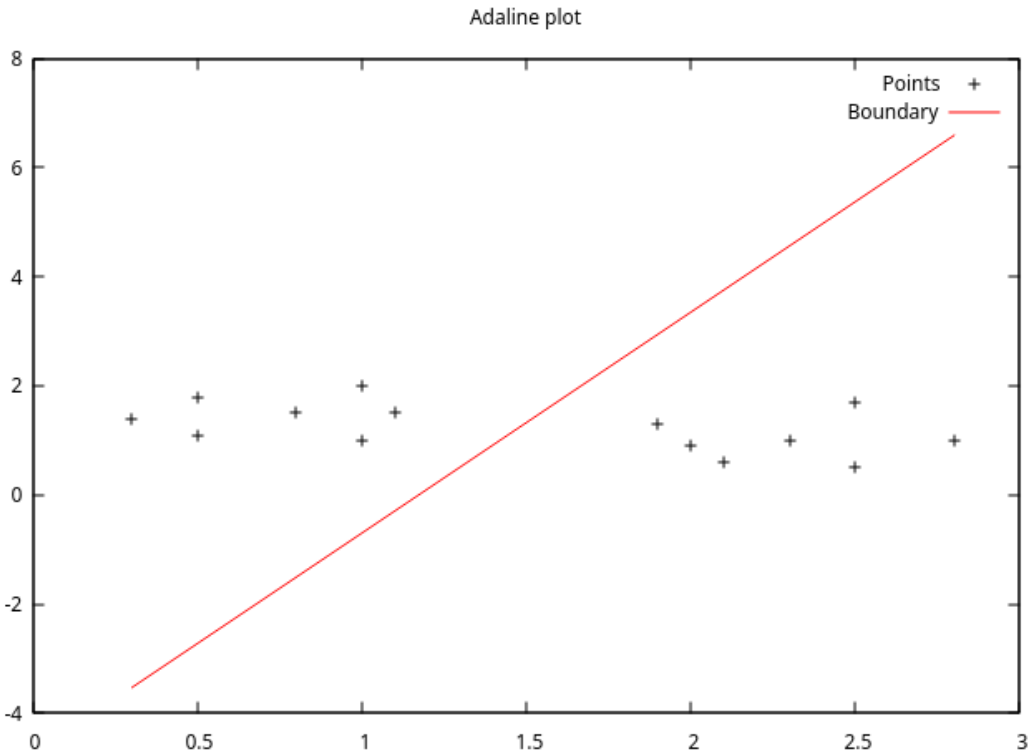
Executando a função com a base de dados, pesos iniciais aleatórios, alpha igual a 0.01 e número de ciclos máximo igual a 1000 foi obtido que os pesos finais são -1.0740 e 0.2652, e o bias final é 1.2602171.

Com os pesos e o bias finais foi possivel começar o processo de plotagem, para a plotagem foi utilizada uma função similar a da utilizada para o perceptron já que a equação da reta e os pontos eram os mesmos:

```
(defun plot (x y)
  (progn
```

```
        (vgplot:plot x y "k+;Points"
                                 '(0.3 2.8)
                                 (list (equation
                                         (adaline table (RANDOM-WEIGHTS)
                                                        TABLE-OUTPUT 0.01 1000) 0.3)
                                         (equation
                                          (adaline table (RANDOM-WEIGHTS)
                                                        TABLE-OUTPUT 0.01 1000) 2.8))
                                 "r-;Boundary")
        (vgplot:title "Adaline plot")))

(defun apply-plot (inputs)
  (plot (map 'list #'first inputs) (map 'list #'second inputs)))
```

Usando essas funções obteve-se o seguinte gráfico:



Por fim, foi hora de plotar o erro quadrático total obtido ao longo do treinamento em função do número de ciclos. Para calcular o erro quadrático total foram usadas as seguintes funções:

```
(defun squared-error (output weights input)
  (expt (- output (yliquid weights input)) 2))
```

```
(defun apply-squared-error (outputs weights inputs)
  (apply-squared-error-aux outputs weights inputs 0))

(defun apply-squared-error-aux (outputs weights inputs result)
  (cond
        ((and (null outputs) (null inputs)) result)
        (t (apply-squared-error-aux
                (rest outputs)
                weights
                (rest inputs)
                (+ result (squared-error (first outputs) weights (first inputs)))))))
```

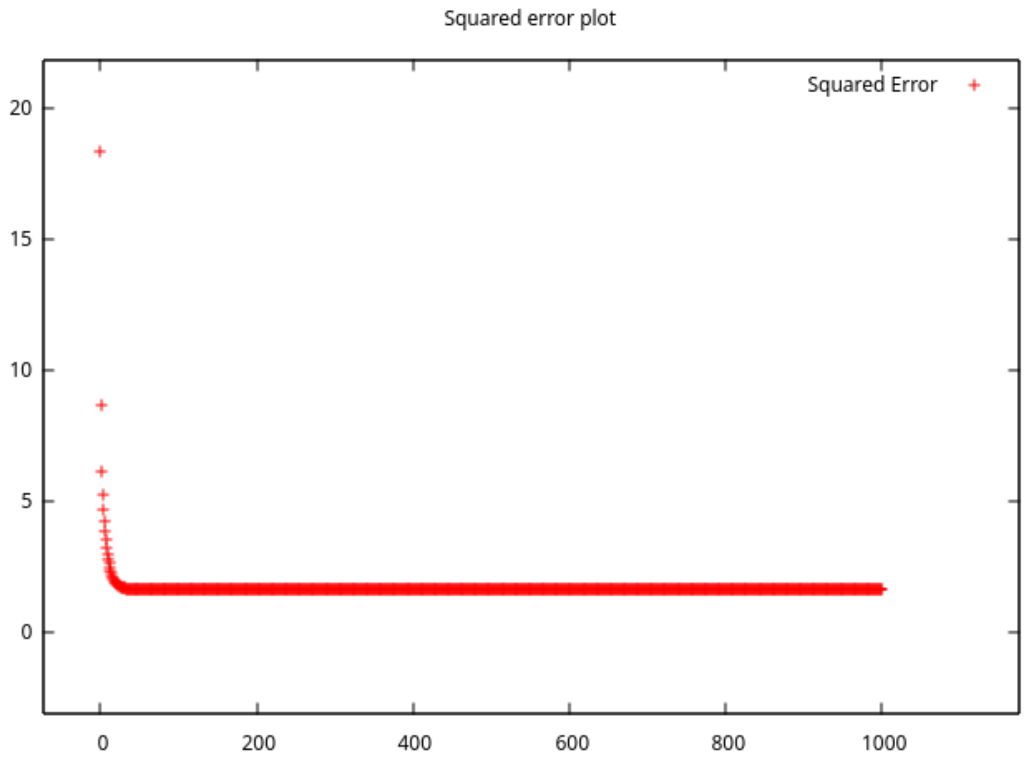
Para obter os valores do erro quadrático no meio do treinamento foi criada uma variação da função de treinamento que retornava uma lista com o número total de ciclos passados e o uma lista com todos os erros quadráticos apresentados nesses ciclos:

```
(defun adaline-plot (inputs weights outputs learning-rate max-cicles)
 (adaline-aux-plot inputs weights outputs learning-rate max-cicles 0 '()))

(defun adaline-aux-plot (inputs weights outputs learning-rate max-cicles cicles error)
  (cond
        ((eql max-cicles 0) (list cicles error))
        (t (adaline-aux-plot
                inputs
                (apply-ajust-weights inputs weights outputs learning-rate)
                outputs
                learning-rate
                (- max-cicles 1)
                (+ cicles 1)
                (append error (list (apply-squared-error outputs weights inputs)))))))
```

Assim, estava tudo pronto para criar a função de plotar o erro quadrático:

```
(defun plot-error (x y)
  (progn
        (vgplot:plot x y "r+; Squared Error")
        (vgplot:title "Squared error plot")))

(defun apply-plot-error (list)
  (plot-error (loop for x upto (first list) collect x) (second list)))
```

Finalmente, ao utilizar essas funções obteve-se o seguinte gráfico:



4 Conclusão

Foi possivel observar por meio da implementação de ambas as redes neurais a evolução das técnicas de treinamento dos neurônios artificiais. Ademais, foi possível observar a implementação da taxa de aprendizado, ou alpha, que no perceptron não produz alterações significativas, mas no adaline é fundamental para o funcionamento. Uma vez que ao alterar os valores desse foi possível observar que para valores muito altos o programa, além de não aprender o comportamento desejado gera pesos tão grandes que quebram o limite do dado float, efetivamente crashando o programa. Sendo assim é preferível que essa seja treinada por mais tempo e obtenha resultados mais precisos.

5 Resultados

5.1 Perceptron

Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1

Weights: [-2.6 2.1999998] b: 1.0

5.2 Adaline

5.2.1 Taxa de aprendizado = 0.01

Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1

Weights: [-1.0740621 0.26515815] b: 1.260376

5.2.2 Taxa de aprendizado = 0.05

Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1

Weights: [-1.1579641 0.1814173] b: 1.324608

5.2.3 Taxa de aprendizado = 0.1

Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1

Weights: [-1.224075 0.057512786] b: 1.4977645

5.2.4 Taxa de aprendizado = 0.25

Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1

Weights: [-1.0327283 0.2637611] b: 0.96545476

5.2.5 Taxa de aprendizado = 0.30

Expected output: 1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1
Expected output: -1 Obtained output: -1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: -1

Weights: [-0.90139055 0.3355062] b: 0.5002496

5.2.6 Taxa de aprendizado = 0.35

Expected output: 1	Obtained output: -1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: -1
Expected output: 1	Obtained output: 1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: -1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: -1
Expected output: -1	Obtained output: -1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: -1
Expected output: -1	Obtained output: -1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: -1

Weights: [-0.3881336 0.49306318] b: -0.271829

5.2.7 Taxa de aprendizado = 0.38

Expected output: 1	Obtained output: 1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: 1
Expected output: 1	Obtained output: 1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: 1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: 1
Expected output: -1	Obtained output: 1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: 1
Expected output: -1	Obtained output: 1
Expected output: 1	Obtained output: 1
Expected output: -1	Obtained output: 1

Weights: [38.831238 26.289228] b: -24.644953

5.2.8 Taxa de aprendizado = 0.39

Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: 1
Expected output: -1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: 1
Expected output: -1 Obtained output: 1
Expected output: 1 Obtained output: 1
Expected output: -1 Obtained output: 1

Weights: [1.2145842e21 6.975444e20] b: -8.594842e20