Relatório 6 - Engenharia de características

1 Introdução

Os resultados desse relatório foram obtidos com base no arquivo sonal.all-data encontrado na base de dados do UCI: `https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/` .

A engenharia de características consiste na transformação de dados brutos em vetores de característica, ou seja, aplicar transformações nos dados brutos a fim de que esses possam ser usados em softwares. Os modelos de aprendizagem de máquina representam as características como vetores de números reais, e usam de estratégias como o one-hot enconding para poderem classificar todas as características dos itens de maneira numérica.

Quando os dados apresentam duas entradas com grandezas muito diferentes é necessário aplicar uma normalização aos dados para que a variação de uma entrada de alta grandeza não ofusque a variação de uma de baixa grandeza.

É normal no treinamento de uma rede neural separar os dados que vão ser utilizados para treiná-la em um conjunto de treinamento e outro de teste, em que, o de treinamento é utilizado, como o nome já diz, para treinar a rede neural e o de teste é utilizado para verificação da acurácia e precisão da rede neural. Apesar de essa ser uma abordagem normal a mais comum é separar um terceiro conjunto, o de avaliação, este será utilizado para verificar se a rede não está sendo super treinada, tornando assim o processo de treinamento mais rápido e eficaz.

Nesse relatório foi utilizada a estratégia de divisão dos dados em três conjuntos e como as grandezas das entradas se mostraram constantes não foi necessário aplicar uma normalização.

2 Objetivo

- Treinar um perceptron com função de ativação logística (sigmóide binária) para classificar sinais de um sonar.

3 Desenvolvimento

Para realizar o treinamento da rede neural foi utilizada a linguagem de programação Common Lisp com um paradigma primordialmente funcional. Como a rede neural em si foi feita como projeto para a semana passada só foi necessário realizar algumas alterações a fim de adaptá-la para o caso atual. A primeira delas foi a alteração da função de ativação, que antes era sigmóide bipolar e agora deve ser sigmóide binária. Para isso a função de calculo da sigmóide e da derivada dela tiveram os nomes e corpos alterados:

```
(defun activation-function (output)
  (- (/ 2 (+ 1 (exp (* -1 output)))) 1))
(defun derivated-activation-function (output)
  (* 1/2 (+ 1 (activation-function output)) (- 1 (activation-function
  ↪  output))))
```

Depois disso foi hora de criar as funções para tratamento da entrada. Como esta veio por meio de um arquivo com 208 linhas de 60 valores separados por virgulas e terminadas com ou a letra R ou M (indicando que os valores são de uma região com rochas ou minas respectivamente) foi necessária a criação de algumas funções para convertê-la em uma lista de valores, embaralhar esses valores e mudar as letras no seu final para 0 no caso de rochas e 1 e adicionar 1 entre esse o valor final das entradas e a indicação:

```
;; Colects all the elements of the base file and transforms they in a
↪  list of lists
;; String -> List of lists
(defun file-to-list (filename)
  (with-open-file (stream filename)
    (loop for line = (read-line stream nil)
       while line
       collect  (ajust-input (mapcar #'string-first (uiop::split-string
       ↪  line :separator '(#\,)))))))

;; Gets the first word from a string and returns it
(defun string-first (string)
  (with-input-from-string (str string) (read str nil nil)))

;; Shuffles the list
;; List -> list
(defun list-shuffle (sequence)
  (loop for i from (length sequence) downto 2
```

```
        do (rotatef (elt sequence (random i))
                    (elt sequence (1- i ))))
  sequence)

;; Ajusts the input so that it is viable for the program
;; List -> list
(defun ajust-input (list)
  (append (append (butlast list) '(1))
          (cond
            ((equal '(R) (last list)) '(0))
            ((equal '(M) (last list)) '(1))
            (t (error "Invalid input")))))
```

Tudo isso foi feito a fim de separar essa entrada tratada em uma lista de entradas utilizáveis e uma lista de targets, para isso foram criadas as duas funções, separated-inputs e separated-targets:

```
(defun separated-inputs (inputs)
  (mapcar #'(lambda (input) (butlast input)) inputs))
(defun separated-targets (inputs)
  (mapcar #'(lambda (input) (first (last input))) inputs))
```

Para facilitar os testes foram criadas as seguintes variáveis:

```
(defvar table (list-shuffle (file-to-list "./data/sonar.all-data")))
(defvar table-in (separated-inputs table))
(defvar table-out (separated-targets table))
```

Para o treinamento foi necessário também separar as entradas e targets em três listas, uma que seria usada para treinamento, outra para validação e uma última que seria usada para os testes, para isso foi criada a seguinte função:

```
;; Recives a list and returns it divided in three porcentages
;; List, float. float. float -> List, list, list
(defun divide-input (list training-percentage evaluation-percentage
↪  test-percentage)
  (if (not (= (apply #'+ (list training-percentage
  ↪  evaluation-percentage test-percentage)) 1))
      (error "The values of the percentages must sum to 1")
      (let* ((size (list-length list)))
```

```
        (values
         (subseq list 0 (round (* size training-percentage)))
         (subseq list
                 (round (* size training-percentage))
                 (round (* size (+ training-percentage
                 ↪  evaluation-percentage))))
         (subseq list
                 (round (* size (+ training-percentage
                 ↪  evaluation-percentage)))
                 (round (* size (+ training-percentage
                 ↪  evaluation-percentage test-percentage)))))))
```

Ainda para que o treinamento fosse feito da maneira correta foi necessário criar uma função para testar a acurácia:

```
;; Tests the accuracy of a set of weights
(defun test-accuracy (inputs targets weights-ij weights-jk)
  (let* ((size (list-length inputs)))
    (* (/ (apply #'+ (mapcar #'
                     (lambda (input target)
                       (if (eql target
                                (round (foward (y-in-k (first
                                ↪  weights-jk) (zj (y-in-j weights-ij
                                ↪  input))))))
                           1 0)) inputs targets)) size) 100)))
```

Com todas essas funções prontas foi hora de alterar as funções de treinamento e teste das condições de parada, nelas foi necessário manter uma variável ac-max que mantêm os valores de saída para os quais obteve-se a maior acurácia e ao fim do treinamento retornar esses valores:

```
;; Trains the weights based on a number of cicles or minimum squared
↪  error
(defun training (inputs eval-list num-hiden-layer learning-rate
↪  max-cicles min-error)
  (let*
      ((weights-jk (random-weights 1 num-hiden-layer))
       (weights-ij (random-weights num-hiden-layer (- (list-length
       ↪  (first inputs)) 1))))
    (test-cicles
     (separated-inputs inputs)
```

```
     (separated-targets inputs)
     eval-list weights-ij weights-jk learning-rate max-cicles min-error
     ↪  0 (list 0) (list weights-ij weights-jk 0 (list 0) 0))))
;; Tests whether stop conditions have been reached
(defun test-cicles (inputs targets eval-list weights-ij weights-jk
↪  learning-rate max-cicles min-error cicles error ac-max)
  (cond
    ((or
      (eql cicles max-cicles)
      (and (<= (first (last error)) min-error) (< 0 cicles)))
     (values (nth 0 ac-max) (nth 1 ac-max) (nth 2 ac-max) (nth 3
     ↪  ac-max) (nth 4 ac-max)))
    (t (multiple-value-bind (w-ij w-jk er)
           (ajust-all-weights inputs targets weights-ij weights-jk
           ↪  learning-rate 0)
         (test-cicles inputs targets eval-list w-ij w-jk learning-rate
         ↪  max-cicles min-error (1+ cicles) (append error (list er))
                      (if (> (first (last ac-max)) (test-accuracy
                      ↪  inputs targets weights-ij weights-jk)) ac-max
                          (list weights-ij weights-jk cicles (rest
                          ↪  error) (test-accuracy inputs targets
                          ↪  weights-ij weights-jk)))))))))
```

Por fim, foi necessário alterar as funções que comparam as saídas com o target, nessas for retirado o print dos pesos, que para 60 entradas poluíam muito a tela e no lugar deles foi colocado um format para printar a acurácia final.

```
;; Compares the output presented by the program and the target
(defun compare-outputs (inputs weights-ij weights-jk)
  (compare-outputs-aux
   (separated-inputs inputs)
   (separated-targets inputs) weights-ij weights-jk (list-length
   ↪  inputs) 0))
(defun compare-outputs-aux (inputs targets weights-ij weights-jk size
↪  hits)
  (cond
    ((null inputs)
     (format t "~%Accuracy: ~2$%" (* (/ hits size) 100)))
    (t (format t "Expected: ~a | Result ~a ~%"
               (first targets)
               (foward (y-in-k (first weights-jk) (zj (y-in-j
               ↪  weights-ij (first inputs))))))))
```

```
(compare-outputs-aux (rest inputs) (rest targets) weights-ij
↪  weights-jk size
                     (if (eql
                          (first targets)
                          (round (foward (y-in-k (first
                          ↪  weights-jk) (zj (y-in-j weights-ij
                          ↪  (first inputs)))))))
                         (1+ hits) hits)))))
```

4 Resultados e discussões

Por meio de diversos testes foi observado que os melhores resultados foram obtidos para uma separação da entradas com 70% para o treinamento, 10% para avaliação e 20% para testes; taxa de aprendizado de 0.1 e erro mínimo como 0.0001. Com esses valores fixos foram alterados o número máximo de ciclos e o número de neurônios na camada escondida a fim de obter a melhor acurácia.

Os resultados obtidos variaram a cada interação graças à variação dos pesos iniciais, a acurácia variou de 78% à 88% mas geralmente se manteve próximos a 83%. Foi observado por meio de testes que um aumento significativo no número de ciclos e no número de neurônios na camada escondida não produziam muita alteração na acurácia final, na verdade, ao aumentar muito esses valores obteve-se uma acurácia em média menor que a encontrada com 5 neurônios na camada escondida e 100 ciclos.

Mesmo que os resultados da acurácia façam parecer que um número menor de ciclos e de neurônios na camada escondida são ideais deve-se atentar-se ao fato de que para o calculo da acurácia foi feito um truncamento dos valores de saída da rede neural, ou seja, mesmo que a acurácia seja grande isso não significa que a precisão dos resultados foi. Como nesse exemplo os resultados são binários isso pôde ser relevado, mas em outras aplicações seria necessário treinar mais a rede neural a fim de aumentar sua precisão.

Seguem abaixo alguns resultados encontrados:

4.1 Número máximo de ciclos = 100 e número de neurônios na camada escondida = 5

Expected: 0 Result 0.12754261
Expected: 1 Result 0.7122468
Expected: 0 Result 0.3600818
Expected: 0 Result -0.37999725
Expected: 0 Result -0.16681194
Expected: 1 Result 0.7266029
Expected: 1 Result 0.98392045
Expected: 1 Result 0.6346991
Expected: 1 Result 0.9569601
Expected: 0 Result 0.17585242
Expected: 1 Result 0.91224706
Expected: 1 Result 0.8936647
Expected: 1 Result 0.9375453
Expected: 1 Result 0.82716703
Expected: 0 Result 0.97899044
Expected: 1 Result 0.9856545
Expected: 1 Result 0.7778399
Expected: 0 Result 0.20178878
Expected: 1 Result 0.97980106
Expected: 0 Result -0.19049889
Expected: 1 Result 0.5646348
Expected: 1 Result 0.9763777
Expected: 0 Result -0.12697923
Expected: 0 Result 0.7189543
Expected: 1 Result 0.6255603
Expected: 0 Result 0.32717466
Expected: 1 Result 0.33842552
Expected: 0 Result 0.7375262
Expected: 1 Result 0.92204785
Expected: 0 Result -0.21108681
Expected: 0 Result 0.48123193
Expected: 0 Result -0.1425345
Expected: 0 Result 0.78712773
Expected: 1 Result 0.9800242
Expected: 0 Result -0.10190028
Expected: 1 Result 0.52587163
Expected: 0 Result 0.05076492
Expected: 1 Result 0.99262273
Expected: 0 Result 0.24110484
Expected: 0 Result 0.4601109
Expected: 1 Result 0.8292868
Expected: 0 Result 0.26734316

Accuracy: 88.10%

4.2 Número máximo de ciclos = 1000 e número de neurônios na camada escondida = 20

Expected: 0 Result 0.29759264
Expected: 1 Result 0.82394445
Expected: 0 Result 0.016212821
Expected: 0 Result -0.48786223
Expected: 0 Result -0.3506111
Expected: 1 Result 0.82444847
Expected: 1 Result 0.9833133
Expected: 1 Result 0.83365345
Expected: 1 Result 0.97672355
Expected: 0 Result -0.17915559
Expected: 1 Result 0.9879695
Expected: 1 Result 0.8037802
Expected: 1 Result 0.9249432
Expected: 1 Result 0.93501854
Expected: 0 Result 0.975657
Expected: 1 Result 0.99631345
Expected: 1 Result 0.6815473
Expected: 0 Result 0.13629127
Expected: 1 Result 0.98864484
Expected: 0 Result -0.23089868
Expected: 1 Result 0.52774847
Expected: 1 Result 0.987844
Expected: 0 Result -0.22981292
Expected: 0 Result 0.20382261
Expected: 1 Result 0.7505274
Expected: 0 Result 0.31851137
Expected: 1 Result 0.49605143
Expected: 0 Result 0.90869963
Expected: 1 Result 0.9699869

Expected: 0 Result -0.20915699
Expected: 0 Result 0.837947
Expected: 0 Result -0.027148664
Expected: 0 Result 0.95129085
Expected: 1 Result 0.9887707
Expected: 0 Result -0.33259785
Expected: 1 Result 0.33012486
Expected: 0 Result 0.16772687
Expected: 1 Result 0.9982511
Expected: 0 Result -0.4619537
Expected: 0 Result 0.27747536
Expected: 1 Result 0.8827714
Expected: 0 Result 0.13099527

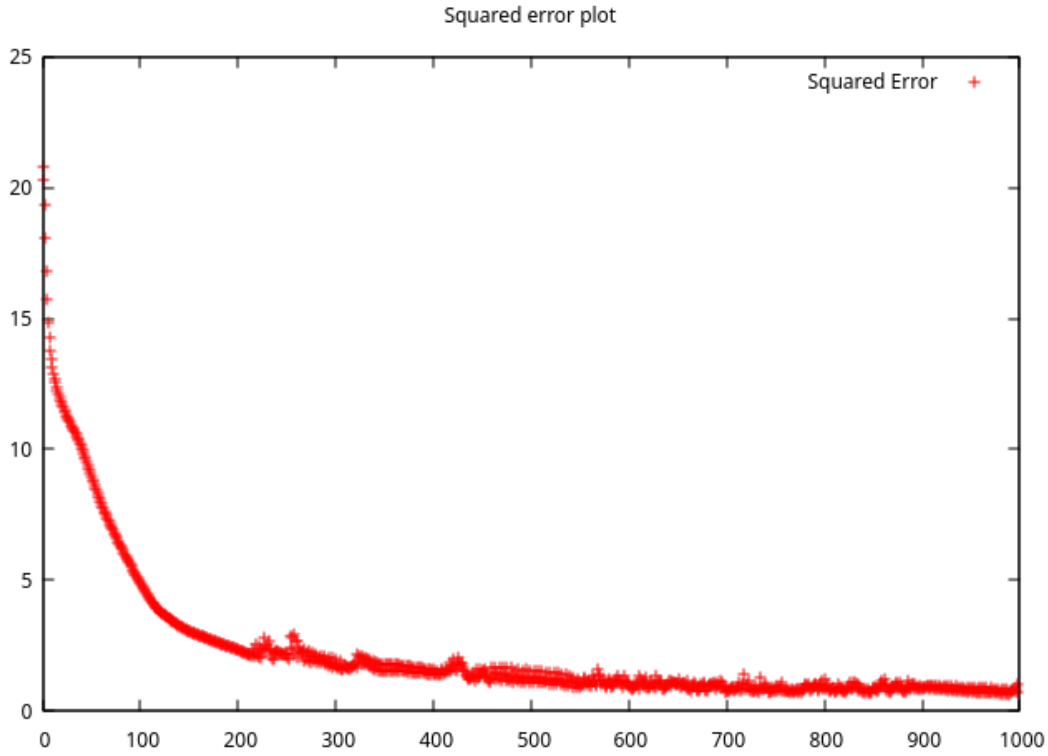
Accuracy: 85.71%

4.3 Número máximo de ciclos = 10000 e número de neurônios na camada escondida = 20

Expected: 0 Result 0.59114504
Expected: 1 Result 0.99921906
Expected: 0 Result -0.22580123
Expected: 0 Result 0.84845567
Expected: 0 Result -0.1333481
Expected: 1 Result 0.995986
Expected: 1 Result 0.99992037
Expected: 1 Result 0.9892739
Expected: 1 Result 0.99990344
Expected: 0 Result -0.054320335
Expected: 1 Result 0.99994826
Expected: 1 Result 0.99811566
Expected: 1 Result 0.9984484
Expected: 1 Result 0.9989722
Expected: 0 Result 0.9997511
Expected: 1 Result 0.9999957
Expected: 1 Result 0.9905716
Expected: 0 Result 0.75644875
Expected: 1 Result 0.9990196
Expected: 0 Result -0.082856536
Expected: 1 Result 0.9186301
Expected: 1 Result 0.999984
Expected: 0 Result -0.0015053749
Expected: 0 Result 0.91182053
Expected: 1 Result 0.99196076
Expected: 0 Result -0.045452952
Expected: 1 Result 0.98752975
Expected: 0 Result 0.99539435
Expected: 1 Result 0.99905634
Expected: 0 Result -0.516452
Expected: 0 Result 0.9982965
Expected: 0 Result 4.1913986e-4
Expected: 0 Result 0.99618113
Expected: 1 Result 0.9999764
Expected: 0 Result 0.20215762
Expected: 1 Result 0.8606305
Expected: 0 Result -0.056747794
Expected: 1 Result 0.9999962
Expected: 0 Result 0.67813075
Expected: 0 Result 0.16862309
Expected: 1 Result 0.9986986
Expected: 0 Result 0.68375266

Accuracy: 73.81%

4.4 Gráfico do erro quadrático para 1000 ciclos e 5 neurônios na camada escondida



5 Conclusão

Em suma, foi possível observar por meio dessa aplicação uma utilização das redes neurais para solução de um problema real, e mesmo que a maior acurácia tenha sido de 88.10% ainda é impressionante o fato de que foi possível treinar um programa para um sonar de maneira tão simples e intuitiva.

Ademais é interessante, ressaltar que na aplicação atual os resultados foram truncados, e por isso mesmo que para um número de ciclos menor tenha-se obtido um valor de acurácia maior, a precisão dos resultados deixou a desejar. Pode-se observar que para um número maior de ciclos a precisão aumenta significativamente.