IA

Cecília Carneiro e Silva, Valks

06/2016

Contents

1 Implementação

Segue abaixo a implementação de uma rede neural em Lush http://lush.sourceforge.net/, mais especificadamente uma 'Convolutional Neural Networks' (ConvNets) .
As diferenças entre elas se resume no fato das ConvNets assumirem claramente que desejam que sua estrada seja uma imagem, com isso, algumas propriedades são 'escondidas'.

O código abaixo executa o clássico problema MNIST, classificação de números manuscritos. Entraremos com imagens e seus respectivos labels, o treinamento será feito com 60000 <imagens, labels>, disponíveis em: http://yann.lecun.com/exdb/mnist/ .

Esse trabalho foi feito com ajuada da biblioteca gblearn2, biblioteca padrão do Lush para 'Machine Learn'.

```
;;MNIST - lush, gblearn
;;Cecília Carneiro e Silva
;;DATE: 06/2016
```

```
(libload "gblearn2/gb-trainers")
(libload "gblearn2/gb-meters")
(libload "gblearn2/net-cscscf")
(libload "gblearn2/demos/dsource-mnist")

(defvar *mnist-path* "/home/cecilia/eblearn-code/demos/mnist/")

(de mnist-main (treino-size)
    (let ((treino-db
           (new dsource-idx3l-narrow
                (new dsource-mnist
                     (load-matrix (concat-fname *mnist-path* "train-images-idx3-ubyte")
                     (load-matrix (concat-fname *mnist-path* "train-labels-idx1-ubyte")
                     32 32 0 0.01) ;;w h bias offset
                treino-size 0))) ;;cria n do dsource-mnist
      (let ((nOut 10)
            (objetivo 1))
        (let ((labels (int-matrix nOut))
              (objetivos (float-matrix nOut nOut)))
          (objetivos ()() (- objetivo))
          (for (i 0 (- nOut 1))
               (objetivos i i objetivo)
               (labels i i))
                                      ;(print labels)
                                      ;(print (objetivos 1 1))
                                      ;(print (objetivos 0 1))
          (let ((treino-parametro (new idx1-ddparam 0 treino-size)))
                                      ;(print treino-parametro)
            (let ((lenet-rede
                   (new-lenet5 32 32
                               5 5
                               2 2
                               5 5
                               2 2
                               200  ;; dim of hidden layer
                               10 treino-parametro)))
                                      ;(print lenet-rede)
              (let ((minha-rede
                     (new idx3-supervised-module
                          lenet-rede
```

```
                    (new edist-cost labels 1 1 objetivos)
                    (new max-classer labels))))
              (let ((treino-rede
                     (new supervised-gradient minha-rede treino-parametro)))
                (let ((treino-medida (new classifier-meter)))
                  (==> :minha-rede:machine forget 1 2) ;;inicializa os pesos da rede
                  (==> treino-rede compute-diaghessian treino-db 200 0.02)
                  (==> treino-rede train treino-db treino-medida 0.5 0)
                  (printf "Rede treinada com %d imagens.\n\n" treino-size)
                  (eval
                   `(lambda(detectar-size &optional (numero-imagem false))
                      (let ((detectar-db
                             (new dsource-idx3l-narrow
                                  (new dsource-mnist
                                       (load-matrix (concat-fname *mnist-path* "t10k
                                       (load-matrix (concat-fname *mnist-path* "t10k
                                       32 32 0 0.01)
                                  detectar-size 0)))
                        (let ((detectar-medida (new classifier-meter)))
                          (let ((detectar-rede ,treino-rede));;dps de treinada
                            (cond ((and (numberp numero-imagem) (< numero-imagem dete
                                   (printf "Testando o elemento %d do arquivo t10k-im
                                   (let ((resultado
                                          (nth (==> detectar-rede
                                                    test-sample detectar-db detectar-
                                               (esperado (mostrarImagem numero-imagem)))
                                     (printf "Esperado = %d   |   Obtido = %d \n" esp
                                     ))
                                  (t (printf "Testando a rede para %d imagens.\n" det
                                     (==> detectar-rede test detectar-db detectar-med
                                     (==> detectar-medida display))
                                  )
                            )
                          )
                        )
                      ))
                   )))
              ))
        ))
  ))
```

```
(de new-lenet5 (image-height
                image-width
                ki0 kj0 si0 sj0 ki1 kj1 si1 sj1
                hid output-size net-param)
    (let ((table0 (full-table 1 20))
          (table1 (full-table 20 50))
          (table2 (full-table 50 hid)))
      (new net-cscscf
           image-height image-width
           ki0 kj0 table0 si0 sj0
           ki1 kj1 table1 si1 sj1
           ;; WARNING: those two numbers must be changed
           ;; when image-height/image-width change
           (/ (- (/ (- image-height (1- ki0)) si0) (1- ki1)) si1)
           (/ (- (/ (- image-width (1- kj0)) sj0) (1- kj1)) sj1)
           table2
           output-size
           net-param)))

(de mostrarImagem (numeroMat)
    (let ((imagemMat (load-matrix (concat-fname *mnist-path* "t10k-images-idx3-ubyte")))
          (labelMat (load-matrix (concat-fname *mnist-path* "t10k-labels-idx1-ubyte"))))
      (print "Imagem:")
      (for (i 0 27)
           (for (j 0 27)
                (let ((intensidade (imagemMat numeroMat i j)))
                  (cond ((< intensidade 10)  (printf "%d    " intensidade))
                        ((< intensidade 100) (printf "%d   "  intensidade))
                        (t                   (printf "%d  "   intensidade))
                        )))
           (printf "\n"))
      (labelMat numeroMat)
      ))
```

2 LeNet-5, convolutional neural networks

Convolutional Neural Networks are are a special kind of multi-layer neural networks. Like almost every other neural networks they are trained with

a version of the back-propagation algorithm. Where they differ is in the architecture. Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing. They can recognize patterns with extreme variability (such as handwritten characters), and with robustness to distortions and simple geometric transformations. LeNet-5 is our latest convolutional network designed for handwritten and machine-printed character recognition.

3 Backpropagation

Com o erro calculado, o algoritmo corrige os pesos em todas as camadas, partindo da saída até a entrada.

4 Gradiente descendente

Método mais comum de minimização de erros, usado no backpropagation.

Basic modules generally do not assume much about the kind of learning algorithm with which they will be trained. The most common form of training is gradient-based training. gradient-based training consists in finding the set of parameters that minimize a particular energy function (generally computed by averaging over a set of training examples).