Neural Network Implementation with MNIST data

Vincent Cognet

20/12/2019

Introduction

Nous entraînerons notre NN sur la base du jeu de test MNIST. The training set contains 60000 examples, and the test set 10000 examples.

Théorie et mathématiques

Soit les 150 observations suivantes représentées par la matrice $X_{150,784}$ (ou tensor 2 dimensions) comprenant 150 lignes pour les 150 observations et 784 colonnes pour les 784 features des observations.

2 matrices de poids $W^1_{32,150}$ et $W^2_{10,150}$ sont utilisées.

- 1er layer de 32 neurones
- 2nd layer de 10 neurones

La sortie $OUTPUT_{150,10}$ est une matrice de 150 lignes avec les 10 colonnes représentant les 10 features que l'on cherche à reconnaître.

Voici le schéma simplifié du NN à 2 couches:

$$X \longrightarrow \otimes W^1 \rightarrow Z^1 \rightarrow \sigma \rightarrow LAYER^1 \longrightarrow \otimes W^2 \rightarrow Z^2 \rightarrow \sigma \rightarrow \hat{Y} \ggg LOSS(\hat{Y}, Y)$$

Calcul matriciel

Cela donne le calcul matriciel ci-dessous:

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,784} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,784} \\ \vdots & \vdots & \ddots & \vdots \\ x_{150,1} & x_{150,2} & \cdots & x_{150,784} \end{pmatrix} \times \begin{pmatrix} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{1,32} \\ w^1_{2,1} & w^1_{2,2} & \cdots & w^1_{2,32} \\ \vdots & \vdots & \ddots & \vdots \\ w^1_{784,1} & w^1_{784,2} & \cdots & w^1_{784,32} \end{pmatrix} = \begin{pmatrix} z^1_{1,1} & z^1_{1,2} & \cdots & z^1_{1,32} \\ z^1_{2,1} & z^1_{2,2} & \cdots & z^1_{2,32} \\ \vdots & \vdots & \ddots & \vdots \\ z^1_{150,1} & z^1_{150,2} & \cdots & z^1_{150,32} \end{pmatrix} \quad (1)$$

$$\sigma(\begin{pmatrix} z^1_{1,1} & z^1_{1,2} & \cdots & z^1_{1,32} \\ z^1_{2,1} & z^1_{2,2} & \cdots & z^1_{2,32} \\ \vdots & \vdots & \ddots & \vdots \\ z^1_{150,1} & z^1_{150,2} & \cdots & z^1_{150,32} \end{pmatrix}) \times \begin{pmatrix} w^2_{1,1} & w^2_{1,2} & \cdots & w^2_{1,10} \\ w^2_{2,1} & w^2_{2,2} & \cdots & w^2_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w^2_{32,1} & w^2_{32,2} & \cdots & w^2_{32,10} \end{pmatrix} = \begin{pmatrix} z^2_{1,1} & z^2_{1,2} & \cdots & z^2_{1,10} \\ z^2_{2,1} & z^2_{2,2} & \cdots & z^2_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ z^2_{150,1} & z^2_{150,2} & \cdots & z^2_{150,10} \end{pmatrix} \quad (2)$$

$$\sigma(\begin{pmatrix} z^2_{1,1} & z^2_{1,2} & \cdots & z^2_{1,10} \\ z^2_{2,1} & z^2_{2,2} & \cdots & z^2_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ z^2_{150,1} & z^2_{150,2} & \cdots & z^2_{150,10} \end{pmatrix}) = \begin{pmatrix} \hat{y}_{1,1} & \hat{y}_{1,2} & \cdots & \hat{y}_{1,10} \\ \hat{y}_{2,1} & \hat{y}_{2,2} & \cdots & \hat{y}_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_{150,1} & \hat{y}_{150,2} & \cdots & \hat{y}_{150,10} \end{pmatrix} \quad (3)$$

$$LOSS(Y, \hat{Y}) = \sum (\begin{pmatrix} \hat{y}_{1,1} & \hat{y}_{1,2} & \cdots & \hat{y}_{1,10} \\ \hat{y}_{2,1} & \hat{y}_{2,2} & \cdots & \hat{y}_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_{150,1} & \hat{y}_{150,2} & \cdots & \hat{y}_{150,10} \end{pmatrix} - \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,10} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,10} \\ \vdots & \vdots & \ddots & \vdots \\ y_{150,1} & y_{150,2} & \cdots & y_{150,10} \end{pmatrix})^2 \quad (4)$$

$$Z_1 = X.W_1 \tag{5}$$
$$LAYER_1 = \sigma(Z_1) \tag{6}$$
$$Z_2 = LAYER_1 * W_2 \tag{7}$$
$$\hat{Y} = \sigma(Z_2) \tag{8}$$
$$LOSS = (\hat{Y} - Y)^2 \tag{9}$$

Calculons la dérivée de la fonction $LOSS$ en fonction de W^2

$$\frac{\delta LOSS}{\delta W_2} = \frac{\delta LOSS}{\delta \hat{Y}} . \frac{\delta \hat{Y}}{\delta Z_2} . \frac{\delta Z_2}{\delta W_2} \tag{10}$$
$$= 2(\hat{Y} - Y).\sigma'(Z_2).LAYER_1 \tag{11}$$

$2(\hat{Y} - Y)$ est une matrice de dimension $(150, 10)$

$\sigma'(Z_2)$ est une matrice de dimension $(150, 10)$

$LAYER_1$ est une matrice de dimension $(150, 32)$

Le calcul matriciel qui sera fait est $t(LAYER_1) * (2(\hat{Y} - Y).\sigma'(Z_2))$, où $*$ est le produit matriciel et . le produit d'Hadamard. Le résultat donne une matrice de dimension $(32, 10)$ qui est de même dimension que W_2

$$t(150, 32) * (150, 10).(150.10) = (32, 150) * (150, 10) = (32, 10)$$

Calculons la dérivée de la fonction $LOSS$ en fonction de W^1

$$\frac{\delta LOSS}{\delta W_1} = \frac{\delta LOSS}{\delta \hat{Y}} . \frac{\delta \hat{Y}}{\delta Z_2} . \frac{\delta Z_2}{\delta LAYER_1} . \frac{\delta LAYER_1}{\delta Z_1} . \frac{\delta Z_1}{\delta W_1} \tag{12}$$
$$= 2(\hat{Y} - Y).\sigma'(Z_2).W_2.\sigma'(Z_1).X \tag{13}$$

$2(\hat{Y} - Y)$ est une matrice de dimension $(150, 10)$

$\sigma'(Z_2)$ est une matrice de dimension $(150, 10)$

W_2 est une matrice de dimension $(32, 10)$

$\sigma'(Z_1)$ est une matrice de dimension $(150, 32)$

X est une matrice de dimension $(150, 784)$

Le calcul matriciel qui sera fait est $t(X) * \{(2(\hat{Y} - Y).\sigma'(Z^2) * t(W^2).\sigma'(Z^1)\}$, où $*$ est le produit matriciel et . le produit d'Hadamard. Le résultat donne une matrice de dimension $(784, 32)$ qui est de même dimension que W_1

$$t(150, 784) * \{(150, 10).(150, 10) * t(32, 10)).(150, 32))\} = (784, 150) * \{(150, 10) * (10, 32).(150, 32)\} \tag{14}$$
$$= (784, 150) * (150, 32) \tag{15}$$
$$= (784, 32) \tag{16}$$

Fonctions d'activation

Pour la fonction d'activation, ici appelée σ, nous utiliserons pour la première couche la fonction $relu(x) = max(o, x)$

Pour la seconde couche, nous utiliserons la fonction sigmoid $f(x) = \frac{1}{1+e^{-x}}$

The R implementation

See below the R implementation,following the good article https://www.r-bloggers.com/how-to-build-your-own-neural-network-

```
# MNIST DATA PREPARATION
library(keras)
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_train <- x_train / 255
y_train <- to_categorical(y_train, 10)

x_test <- mnist$test$x
y_test <- mnist$test$y
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_test <- x_test / 255


my_nn <- list(
  input = x_train,
  weights1 = matrix(runif(32*784),nrow = 784, ncol=32),
  layer1 = matrix(rep(0,32*150), nrow = 150 , ncol= 32),
  weights2 = matrix(runif(32*10),nrow=32, ncol = 10),
  z2 = matrix(rep(0,150*10), nrow = 150, ncol= 10),
  y = y_train,
  output = matrix(rep(0, times = 10*150),nrow = 150, ncol = 10)
)

# the activation function
sigmoid <- function(x) {
  1.0 / (1.0 + exp(-x))
}


# the derivative of the activation function
sigmoid_derivative <- function(x) {
  sigmoid(x) * (1.0 - sigmoid(x))
}

# the relu function and its derivative
relu <- function(x) {
  if (x<=0) 0 else x
}

relumatrix <- function(m) {
  apply(m, c(1,2), relu)
}

relu_derivative <- function(x) {
  if (x<=0) 0 else 1
}

relumatrix_derivative <- function(m) {
  apply(m, c(1,2), relu_derivative)
```

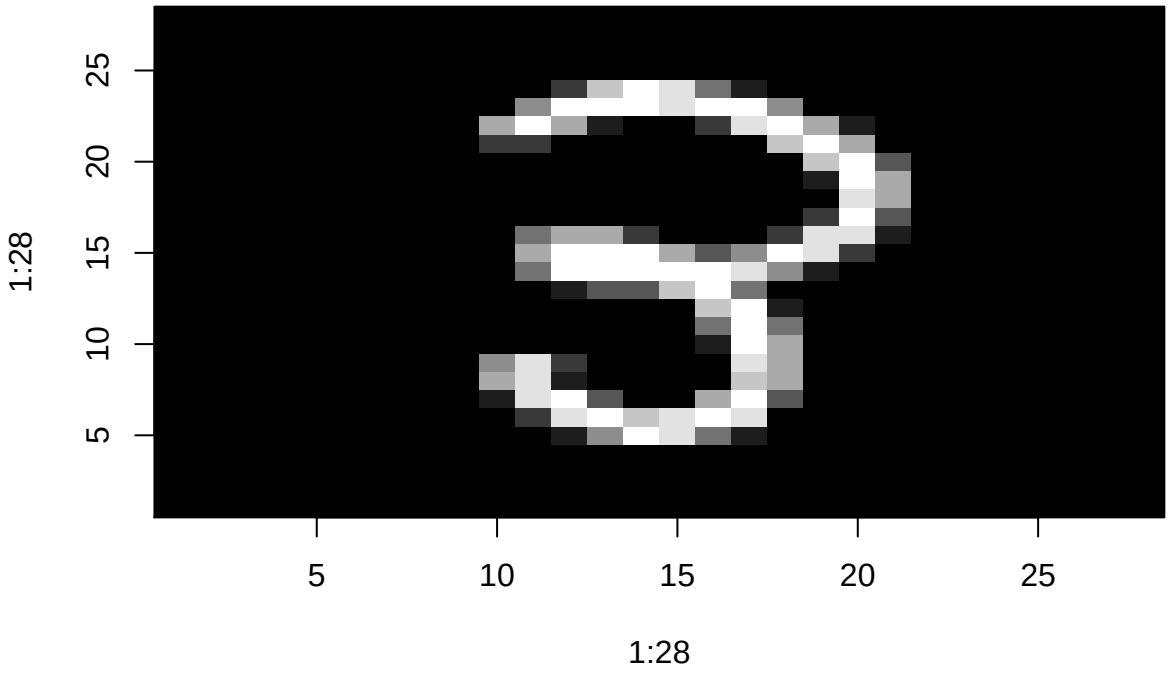
```
}


loss_function <- function(nn, b) {
  sum((nn$y[(((b-1)*batch_size+1):(b*batch_size)),] - nn$output) ^ 2)
}

# batch_size. doit être un multiple du sample en input
batch_size <- 150

feedforward <- function(nn, b) {
  nn$z1 <- nn$input[(((b-1)*batch_size+1):(b*batch_size)),] %*% nn$weights1
  nn$layer1 <- relumatrix(nn$z1)
  nn$z2 <- nn$layer1 %*% nn$weights2
  nn$output <- sigmoid(nn$z2)
  nn
}

image(1:28, 1:28, t(mnist$test$x[311,,]), col=gray((0:255)/255))
```




```
backprop <- function(nn, b, learning_rate) {
  d_weights2 <- (  t(nn$layer1) %*% (2 * (nn$y[(((b-1)*batch_size+1):(b*batch_size)),] - nn$output) *  

  d_weights1 <- ( 2 * (nn$y[(((b-1)*batch_size+1):(b*batch_size)),] - nn$output) * sigmoid_derivative(nn
  d_weights1 <- d_weights1 * relumatrix_derivative(nn$z1)
  d_weights1 <- t(nn$input[(((b-1)*batch_size+1):(b*batch_size)),]) %*% d_weights1
```

```
  # update the weights
  nn$weights1 <- nn$weights1 + (d_weights1 * learning_rate)
  nn$weights2 <- nn$weights2 + (d_weights2 * learning_rate)

  nn
}

# number of times to perform feedforward and backpropagation, and the learning rate
n <- 10
learning_rate <- 1.0

loss_df <- matrix(ncol=nrow(my_nn$input)/batch_size, nrow = n)


test_nn <- function(nn, x) {
  resultats <- sigmoid(relumatrix(x %*% nn$weights1) %*% nn$weights2)
  sortie <-  max.col(resultats) - 1
  comparaison <- matrix(data = c(y_test, sortie), byrow= FALSE, ncol=2)
  comparaison <- cbind(comparaison, comparaison[,1]== comparaison[,2])
  return(colSums(comparaison)[3]/length(x[,1]))
  }


for (i in (1:n)) {
  for (b in (1:(nrow(my_nn$input)/batch_size)))
  {
   my_nn <- feedforward(my_nn, b)
   my_nn <- backprop(my_nn, b, learning_rate)
   loss_df[i,b] <- loss_function(my_nn, b)
  }
  print("epoch"); print(i)
  print(test_nn(my_nn, x_test))
}
```

```
## [1] "epoch"
## [1] 1
## [1] 0.0992
## [1] "epoch"
## [1] 2
## [1] 0.0942
## [1] "epoch"
## [1] 3
## [1] 0.1012
## [1] "epoch"
## [1] 4
## [1] 0.0975
## [1] "epoch"
## [1] 5
## [1] 0.0996
## [1] "epoch"
## [1] 6
## [1] 0.0986
## [1] "epoch"
```

```
## [1] 7
## [1] 0.1027
## [1] "epoch"
## [1] 8
## [1] 0.1029
## [1] "epoch"
## [1] 9
## [1] 0.0979
## [1] "epoch"
## [1] 10
## [1] 0.0987
```

Après apprentissage, validons la performance de notre NN sur un autre jeu de test.

```
test_nn(my_nn, x_test)
```

```
## [1] 0.1
```