Ejercicios de MiniFlow

Efren López Jiménez

I. Introducción

MiniFlow es un marco minimalista de redes neuronales creado por el equipo Udacity Self Driving Car para enseñar los conceptos básicos de aprendizaje profundo.

II. Desarrollo

A. Programa 5

En este programa se modificó la clase **def forward (self):**

```
class Add(Node):
def __init__(self, x, y):
# You could access `x` and `y`
#in forward with
# self.inbound_nodes[0] (`x`) and
#self.inbound_nodes[1] (`y`)
Node.__init__(self, [x, y])

def forward(self):
"""
Set the value of this node (`self.value`)
to the sum of its inbound_nodes.
Your code here!
"""
value = self.inbound_nodes[0].value +
self.inbound_nodes[1].value

self.value = value
```

B. Programa 7

Para este programa no fue necesario modificar nada, ya que la salida esperada de la red mediante la ecuación es la esperada:

$$\sum_i = x_i w_i + b$$

```
class Linear(Node):
def __init__(self,inputs,weights,bias):
Node.__init__(self,[inputs,weights,bias])

# NOTE: The weights and bias properties
#here are not
# numbers, but rather references
#to other nodes.
# The weight and bias values are
#stored within the respective nodes.

def forward(self):
"""
Set self.value to the value of the
linear function output.
Your code goes here!
"""
dot = np.dot(self.inbound_nodes[0].value,
      self.inbound_nodes[1].value)
      self.value =
      dot + self.inbound_nodes[2].value
```

C. Programa 8

Para este programa se realizo la red neuronal pero con matrices, en donde X,W,b son vectores, en este caso ocupé la misma sintaxis del programa 7 ya que para Python nos permite con una sintaxis simple multiplicar una matriz con otra.

```
class Linear(Node):
def __init__(self, X, W, b):
# Notice the ordering of the input
 nodes passed to the
# Node constructor.
Node.__init__(self, [X, W, b])

def forward(self):
"""
Set the value of this node to the
 linear transform output.

Your code goes here!
"""
#X=[0], W=[1], B=[2]
producto =
np.dot (self.inbound_nodes[0].value,
    self.inbound_nodes[1].value)
self.value = producto +
self.inbound_nodes[2].value
```

D. Programa 9

Para este programa fue necesario modificar en dos partes el programa, la primera modificación fue en la función $_simoid$ para que nos retornara la función de la sigmoide

```
 def _sigmoid(self, x):
 """
 This method is separate from `forward`
 because it
 will be used later with `backward` as well.

 `x`: A numpy array-like object.

 Return the result of the sigmoid function.
```

jmzelectronica@gmail.com

```
Your code here!
"""
return 1/(1+np.exp(-x))
#Funcion de la sigmoide


def forward(self):
"""
Set the value of this node to the
result of the
sigmoid function, `_sigmoid`.

Your code here!
"""
# This is a dummy value to prevent
 numpy errors
# if you test without changing this method.
self.value =
 self._sigmoid(self.inbound_nodes[0].value)
#sigmoide = 1+(self.value)
```

E. Programa 10

Para este programa se calculó el error cuadrático medio mediante la siguiente formula

$$C(w,b) = \frac{1}{m}\sum_i ||y(x) - a||$$

```
def forward(self):
"""
Calculates the mean squared error.
"""
# NOTE: We reshape these to avoid
 possible matrix/vector broadcast
# errors.
#
# For example, if we subtract an
 array of shape (3,) from an array
 of shape
# (3,1) we get an array of
shape(3,3) as the result when we want
# an array of shape (3,1) instead.
#
# Making both arrays (3,1)
insures the result is (3,1) and does
# an elementwise subtraction as
expected.
y = self.inbound_nodes[0].value.
reshape(-1, 1)
a = self.inbound_nodes[1].value.
reshape(-1, 1)
# TODO: your code here
m = len[y]
mse = sum((y-a)**2)
c = 1/m*mse
self.value=c
```

F. Programa 12

Par este programa se requiere calcular el gradiente descendiente: Para este caso se modificó el programa en la función $gradient_descent_update$ en donde se calcula la actualización del gradiente, y por otro lado podemos probar con valores en el aprendizaje cercanos a 1.

```
############################
learning_rate = .1 #variando el
#valor de aprenizaje
##############################33
def gradient_descent_update
(x, gradx, learning_rate):
"""
Performs a gradient descent update.
"""
# TODO: Implement gradient descent.

# Return the new value for x
actual = gradx*learning_rate
x = x - actual
return x
```

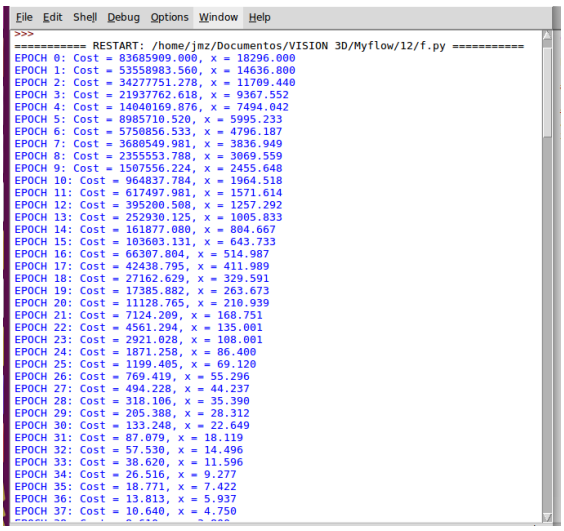

Fig. 1. Épocas en gradiente descendiente

G. Programa 13

Para este programa se pide calcular el gradiente usando la función sigmoide

```
def backward(self):
"""
Calculates the gradient using the derivative
of the sigmoid function.
"""
# Initialize the gradients to 0.
self.gradients =

{n: np.zeros_like(n.value)
```

```
for n in self.inbound_nodes}

# Cycle through the outputs.
#The gradient will change depending
# on each output, so the gradients are
#summed over all outputs.
for n in self.outbound_nodes:
# Get the partial of the cost
#with respect to this node.
grad_cost = n.gradients[self]
"""
TODO: Your code goes here!

Set the gradients property to the gradients
with respect to each input.

NOTE: See the Linear node and MSE node
for examples.
"""
self.gradients[self.inbound_nodes[0]] +=
self.value*(1-self.value) * grad_cost
class MSE(Node):
```

H. Ejercicio del libro

```
#Tarea dos
import numpy as np
import matplotlib.pyplot as plt
import math
import csv
import time
x = np.array([17.3,19.3,19.5,19.7,22.9,23.1,
26.4,26.8,27.6,28.1,28.2,28.7,29,29.6,
29.9,29.9,30.3,31.3,36,39.5,40.4,44.3,44.6,
50.4,55.9])
y = np.array([71.7,48.3,88.3,75,91.7,100,
73.3,65,75,88.3,68.3,96.7,76.7,78.3,
60,71.7,85,85,88.3,100,100,100,91.7,100,
71.7])

xy= x*y
x2= x**2
y2=y*y
a=sum(x)
b=sum(y)
c=sum(xy)
d=sum(x2)
e=sum(y2)
n=25
promx =a/n
promy =b/n
print a,b,c,d

b0 = ((n*c)-(a*b))/((n*d)-(a*a))
b1 = ((d*b)-(a*c))/((n*d)-(a**2))

m = (a*b-n*c)/(a**2-n*d)

b = promy - m*promx


ygorrito = b1 + (b0*30)
print b0
print b1
print ygorrito

linea=[b1,105]
#plt.scatter(x,y)
#plt.plot(linea)
plt.plot(x,y,'o', label='Datos')
plt.plot(x,m*x+b, label='ajuste')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=4)
plt.show()
```

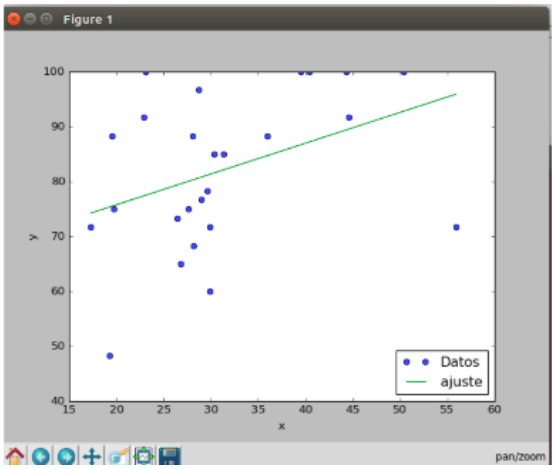

Fig. 2. Regresión Lineal

III. Conclusión

Estos ejercicios ayudan a entender los conceptos sobre la aplicación e implementación de redes neuronales, ademas de el uso de TensorFLow

The authors would like to thank...

IV. Referencias

MiniFLow