Introduction to Perceptron

using python.

Iván Andrés Trujilllo Abella

Facultad de Ingenieria
Pontificia Universidad Javeriana

trujilloiv@javeriana.edu.co
addajaveriana

Background classification problem

Fisher(1936) proposes the linear Discriminant, the problem consist in predict a binary outcome according to a set of features, for instance find the variables that could predict the bankruptcy.
The main objective is construct a $z = \mathrm{w}^t\mathrm{x}$ score whose indicate the probability of belonging to a class.

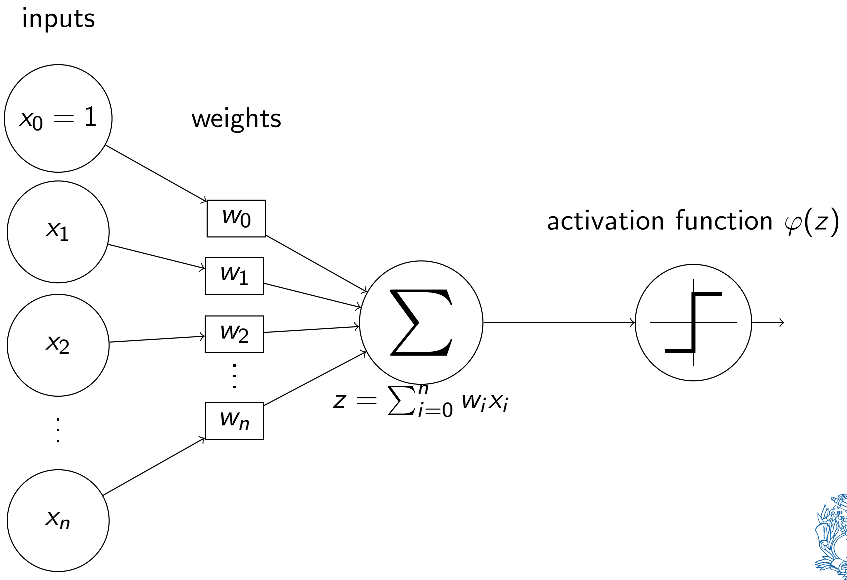
Binary classification problem

$y_0, y_1 \in Y$ and $y_0 \cup y_1 = Y$ and $y_0 \cap y_1 = \emptyset$ both class are exhaustive and are defined without ambiguity. A vector of weights and a vector of features for a

$$w^t x = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} = \sum_{i=1}^{n} w_i x_i = z_i$$

Perceptron

Rossenblant(1958) We can define a vector input x and a vector of weights w and a activation function φ that take as input the inner product of both vectors defined previously $\varphi(w^t x)$.


inputs
$x_0 = 1$
x_1
x_2
$\vdots$
x_n
weights
w_0
w_1
w_2
$\vdots$
w_n
$\sum$
$z = \sum_{i=0}^{n} w_i x_i$
activation function $\varphi(z)$

Activation function

$\varphi()$ could be defined as the sigmoid function.

$$\begin{aligned} p(y=1) &= \varphi(z) \\ p(y=0) &= 1-\varphi(z) \end{aligned} \tag{1}$$

Percepton works with a step function:

$$\varphi(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

Returning a binary outcome.

What set of w values we must choose

Cost function

The cost function could be defined in a soft or a hard way.

$$J(w)_{hard} = \sum_{i=1}^{n} = \max(-y_i \hat{y}, 0) \tag{2}$$

J only count the number of mismatches. However this function not is differentiable.

$$J(w)_{soft} = \sum_{i=1}^{n} = \max(-y_i z_i, 0) \tag{3}$$

if $y_i z_i < 0$ then lost function > 0
if $y_i z_i > 0$ the lost function $= 0$.

Insights

The update of weights is according to the data bias or mistakes, however when the model match to the class then

$$\Delta w_i = (y_i - \hat{y}_i) = 0 \tag{4}$$

where y_i is the real observed data, and $\hat{y}_i$ is the predicted class.
when the $y_i = -1$ and $\hat{y} = 1$ then $\Delta w = -2$, in otherwise $y_i = 1$ and $\hat{y}_i = -1$ then $\Delta w = 2$. in summary:

$$\Delta w_i = \begin{cases} 0 & y_i = \hat{y}_i \\ -2 & y_i < \hat{y}_i \\ 2 & y_i > \hat{y}_i \end{cases}$$

insights

Then when there are mistakes

$$\varphi(w_{i+1}^t \mathrm{x_i}) = \varphi((w_i + \Delta w_i)^t \mathrm{x_i}) = y_i \tag{5}$$

This mean that weights for the vector of features of the sample *i* are update to predict the correct class.

$$w_{i+1} = w_i + \eta \Delta w_i \mathrm{x_i} \tag{6}$$

Perceptron algorithm

```
initialize w:
for each x in sample :
estimate y(x)
w = w + update(w)
```

LAB

Perceptron implementation from scratch

Perceptron from scratch(click here)

sklearn

It is open source library, integrated with scipy and numpy. It is one of the most popular machine learning library on Github.

- Classification (Neural networks Support Vector Machine)
- Decision trees
- Cluster
- Regression

sklearn

```
from skelearn.linear_model import Perceptron
model = Perceptron(penalty=None , max_iter=1000, eta0=0.4,
    random_state=1)
model.fit(X,y)
model.score(X,y) # Print the number of matches
from sklearn.metrics import confusion_matrix
confusion_matrix(y, model.predict(X))
print(model.coef_)
```

Insights more deeply about perceptron

The function *sing* is defined as $\mathbb{R} :\longrightarrow \{-1, 1, 0\}$

Gradient descendent

To talk about more deeply about gradient we need talk about exploding gradient problem
Assume the following convex function :

$$ax^2 + bx + c \tag{7}$$

for practical examples and guarantee a minimum global point we said that $a = 2, b = -3, c = 5$.

$$\begin{aligned} f^1 &= 2ax + b = 0 \\ x^* &= \frac{-b}{2a} \end{aligned} \tag{8}$$

Therefore the minimum point is obtained in $x^* = 0.3$.

Gradient descent

```
import numpy as np
import matplotlib.pyplot as plt
def quadratic(a,b,c,x ):
  return a*x**2 + b*x + c
def Dxquadratic(a,b,x):
  return 2*a*x + b
def Dxxquadratic(a,x):
  return a*x
def dx0quadratic(a,b):
  return (-b)/(2*a) # take in mind the left ritgh precedence
def evaldx0(a,b,c):
  point = dx0quadratic(a,b)
  return quadratic(a,b,c,point)
```

Gradient descent

```
def GDS_Quadratic( x0, learning_rate=0.01, iterations_max=100,
    error_max = 0.00001, a=2,b=-3):
  gradient = Dxquadratic # We defined previously
  xi = x0
  iters = 0
  error = 100
  while (iters < iterations_max) and (error > error_max):
    xj = xi - learning_rate * gradient(a,b, xi)
    error = abs(xi-xj)
    xi = xj
    iters += 1
  return xj,iters
GDS_Quadratic(100, learning_rate=0.3)
```

Why GDS is the steepest ascent

We need some mathematical considerations: for this exercise we can make illustrate the behavior of this 3-D function.

Adaline - Perceptron

MLP

This is a a good way to initialize in this field.
is a matrix

$$\mathcal{A}\vec{x}$$

Recurrent Neural Networks

named entity $X^{(i)<t>}$ the $i-th$ training observation, $T_x^{(i)}$ is the length of input.
one-hot codification to vectors, then we can get the following

x**: in the input appear the** $word_1$ **first after** $word_2$ **and finally** $word_3$**.**

$$Dictionary = \begin{pmatrix} Word_1 \\ word_2 \\ \vdots \\ word_n \end{pmatrix}$$

We can follow the $T_x^{(i)} = 9$.

Encoding the sentence

$|x| = 9$ therefore must be $x^1, x^2, ..., x^9$ encoded vectors. indexing in one-hot sense x $|Dictonary| = n$ for this case, then we have that:

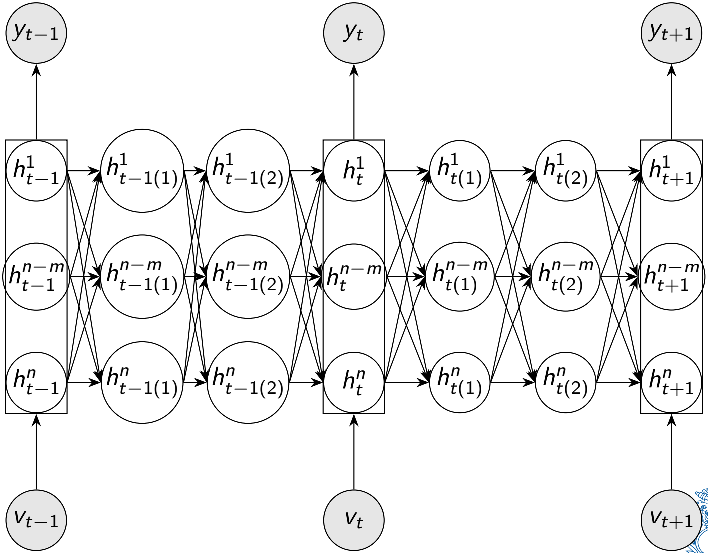
$$x_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, x2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, x_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} .....$$

remember that $|x_i| = |Dictionary|$.

Problems with standard neural networks

a problem of RNN is that not uses information of posteior inputs.

. . .


y_{t-1}
y_t
y_{t+1}
h^1_{t-1}
h^{n-m}_{t-1}
h^n_{t-1}
$h^1_{t-1(1)}$
$h^{n-m}_{t-1(1)}$
$h^n_{t-1(1)}$
$h^1_{t-1(2)}$
$h^{n-m}_{t-1(2)}$
$h^n_{t-1(2)}$
h^1_t
h^{n-m}_t
h^n_t
$h^1_{t(1)}$
$h^{n-m}_{t(1)}$
$h^n_{t(1)}$
$h^1_{t(2)}$
$h^{n-m}_{t(2)}$
$h^n_{t(2)}$
h^1_{t+1}
h^{n-m}_{t+1}
h^n_{t+1}
v_{t-1}
v_t
v_{t+1}

.

Forward propagation

Backward propagation

to research standard logistic regression loss

Different architectures according to the size of input-output

Sentiment analysis only produce a integer output whereas the inputs could be different.
Another example is machine translation.

Sentiment analysis

many-to-one.

Information about presentation

Cada estudiante debe tener entre 3 y 7 articulos.
articulos cortos (máximo 6 páginas) IEEE 0 ACM. max 12 springer.
publicados en los últimos 5 años.

Pontificia Universidad
JAVERIANA
Bogotá

Language model

Softmax

Sampling novel sequences

RNN drawbacks

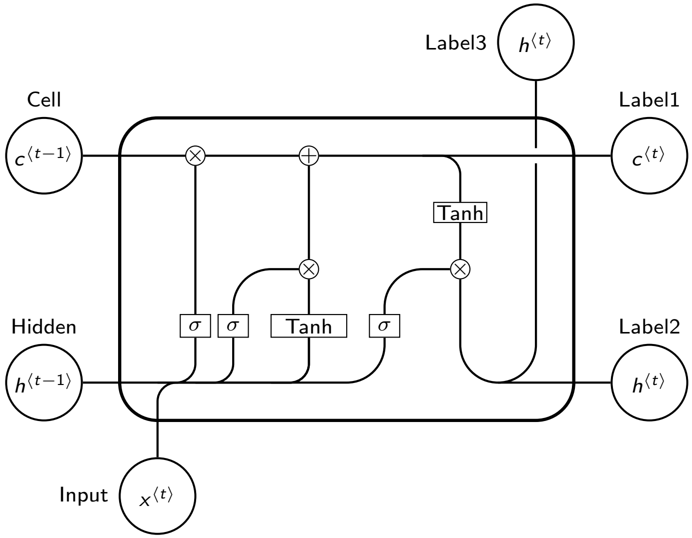
Not capturing long run dependencies

Gated Recurrent Unit (GRU)

tanh activation

Long short-term memory

Recurrent neural networks to time prediction


Label3
$h^{\langle t \rangle}$
Cell
$c^{\langle t-1 \rangle}$
Label1
$c^{\langle t \rangle}$
Tanh
Hidden
σ
σ
Tanh
σ
Label2
$h^{\langle t-1 \rangle}$
$h^{\langle t \rangle}$
Input
$x^{\langle t \rangle}$

Time series prediction

text

Training, validation and test period

here tex

roll forward partitioning

here text

Performance

it also evaluated with some metrics, such as

$$MSE = \frac{1}{n}\sum(\hat{y} - y)^2$$
$$RMSE = \sqrt{MSE}$$
$$MAE = \frac{1}{n}\sum|\hat{y} - y|$$

Moving average

a

RELU

Activation function

Defined as $f(x) = max(0, x)$ is related with softplus. used in CNN due images no require negative numbers and using tanh or sigmoid we can lost information.

softmax

idea: relative weights, remember kolgomorov assumptions: for instance

$$p(j) = \frac{x_j}{\sum x_i}$$

In softmax we have:

$$p(j) = \frac{e^{x_j}}{e^{\sum x_i}}$$

why uses euler number? Note that add a constant not affect probabilities, remember that $e^{a+c} = e^a e^c$.

rate of change $\frac{\partial p(j)}{\partial x_i} = \frac{e^{x_i} \sum e^{x_j} - (e^{x_i})^2}{(\sum e^{x_j})^2} = p(j) - p(j)^2 = p(j)(1 - p(j))$

softmax

what is the $\arg\max_{p(j)} \frac{\partial p(j)}{\partial x_i}$?.

```
## Recurrent neural networks and LSTM
def maxarg_f(pj):
  return pj*(1-pj)
values = [(maxarg_f(pj),pj) for pj in np.linspace(0,1,num=100)]
from operator import itemgetter
max(values,key=itemgetter(0))
# How we can write our own solution?
```

tanh function

is a activation function defined as

$$\begin{aligned} tanh(x) &= \frac{sinh(x)}{cosh(x)} \\ tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \end{aligned} \tag{9}$$

Article

Article

Time series or RNN-LSTM

VAR model is a classic model, that have a good performance in terms of explanability significance statistics.

Images

Digital images are stored in computers, means that are a sequence of numbers. Pixel(picture element; a square) how work? in 1957 russel kirsch digitized the first image.

255	1	0	1	0
1	255	1	1	1
1	0	255	0	0
1	255	1	255	255

RGB

Red, Green and Blue

triplets of these values composed a pixel. (255,255,255), (11111111,11111111,11111111) in binary (255 is the maximum number that 8 bits can represent), therefore form 0 to 255 there are 256 different intensities.

VAR model

Other numerical systems to decimal

$$\sum_{i=0} a_i r^i + \sum_{j=-1} a_j r^j \tag{10}$$

Where r is the maximun digits of another system for instance 2 for binary (0,1). the i index correspond to the digits before point and j after the point. for instance convert 101

$$1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 5. \tag{11}$$

Python implementation

```python
def to_decimal(number, base):
  number = str(number)
  decimal, exp = 0, len(number)-1
  for num in number:
    decimal = decimal + int(num) * (base**exp)
    exp = exp - 1
  return decimal
```

Note that here we uses *int*(*num*) how we can write a own function?

numpy

```
np.zeros((rows,columns)) # return a array with zeros in the
    shape rows,columns.
np.around()
```

Linear regression

Gradient descent

import numpy as np import matplotlib.pyplot as plt
Quadratic Function: def quadratic(a,b,c,x): return a*x**2 + b*x + c
def Dxquadratic(a,b,x): return 2*a*x + b
def Dxxquadratic(a,x): return a*x
def dx0quadratic(a,b): return (-b)/(2*a) take in mind the left ritgh precedence
def evaldx0(a,b,c): point = dx0quadratic(a,b) return quadratic(a,b,c,point)

Gradient descent

idea: small steps near to zero (optimal value) big steps far away from zero. think in the following equation

$$\hat{y}_i = \beta_0 + \beta_1 + u_i \tag{12}$$

now the cost function will be:

$$\sum (y_i - \hat{y}_i)^2 \tag{13}$$

```
Step = Slope * LearningRate
betaZero += Step
```

How change with

Regression line, GD and OLS

Gradient descend is a iterative algorithm that allow us find β_0, β_1 however OLS not is iterative.

Gradient descent is very sensitive to learning rate.

GD algorithm

- derivates of loss function(gradient).
- choose random initial values for parameters
- evaluate parameters in derivates
- calculate step = slope * learningRate
- update parameters (parameter - step)

GDS

buen trabjo

newton method

Logistic regression

General idea in a neural network

Initialize weights... pass the X vector to Network, compare the value with the target and update the weights with the goal of minimize a loss function.

Multilayer Perceptron

Suppose that we have n inputs, and the following layer have m neurons, therefore there will be a matrix $\boldsymbol{W}_{n*m}$ dimensions. Remember that each n input is linked with m neuron.

$$\boldsymbol{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1m} \\ w_{12} & w_{22} & w_{23} & \dots & w_{2m} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nm} \end{pmatrix}$$

insights about neural networks

- trade off small learning rate imply more epochs

mlp

problemas no lienales XOR?

Learning rate decay

it is important for instance implementing mini batch gradient descent, in near values to minimal point if you uses a fixed alpha maybe not converge.

$$\alpha = \frac{1}{1 + r * epochs}\alpha_0 \tag{14}$$

Note that rate decay is another hyperparameter.

Tuning learning rate

Remember that we can relate the learning rate with the loss function, if we plot both we can see that learning rates could no affect the loss, otherwise some values could diverge, this principle could be used to tuning the learning rate.

Keras call backs

Batch and mini batch

Remember that in SGD we uses all samples in data to update the gradient this process is known as *full batch learning*, otherwise *mini batch learning* is a sample that return a approximation to the real gradient.

Momentum

could be a lot of oscillations to reach a minimal point of loss function, however if we take a exponential mean of previous gradients could be get a better performance:

Momentum

we need define a $\lambda \in [0, 1]$ here the subscript indicate the iteration number.

$$
\begin{aligned}
M\boldsymbol{\beta}_0 &= 0 \\
M\boldsymbol{\beta}_1 &= \lambda M\boldsymbol{\beta}_1 + (1 - \lambda)\frac{\partial loss_1}{\partial \boldsymbol{\beta}}
\end{aligned} \tag{15}
$$

Note that for instance $loss_1$ in the epoch one. this formulas are easy generalized for the number of epochs.
to update the parameters then we have

$$\boldsymbol{\beta_j} = \boldsymbol{\beta_i} - \alpha M\boldsymbol{\beta_j} \tag{16}$$

How many gradients we take in account

we can prove why is a average? and how we can prove to where tend the limit?

$$memory = \frac{1}{1-\lambda} \tag{17}$$

Hadamard product

Element wise product

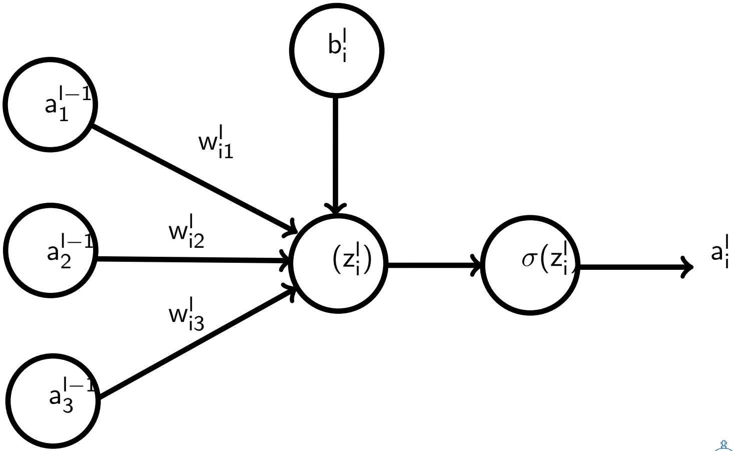
The matrix $\boldsymbol{A} = [a_{ij}]$ and $\boldsymbol{B} = [b_{ij}]$ both of the same dimension. The hadamard product will be $\boldsymbol{A} \oslash \boldsymbol{B} = [a_{ij} b_{ij}]$. think in mind that the identity matrix here is a matrix in which case $\boldsymbol{I} = [i_{ij}] = 1, \forall i, j$. The inverse matrix of $\boldsymbol{A} = [a_{ij}]$ will be denote as $\boldsymbol{A} = [a_{ij}^{-1}]$.

MLP

Minsky y papert

MIP

tackle the problem of linearity.


b_i^l
a_1^{l-1}
w_{i1}^l
a_2^{l-1}
w_{i2}^l
(z_i^l)
$\sigma(z_i^l)$
a_i^l
w_{i3}^l
a_3^{l-1}

Output

$$a_i^l = \sigma(\sum_i w_{ik}^l a_i^{l-1} + b_i^l) \tag{18}$$

when l means the number of layer and i the node. we can write $a_i^l = \sigma(z_i^l)$.

Regla delta generalizada

Jensen inequality

Cross entropy