Back propagation algorithm

María Camila Vásquez Correa

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.model_selection import KFold
```

Multilayer perceptron

The class model receives a list of layers (with its respective activation function) along with the loss tl initialize.

- The fit method performs the training on X and y using learning rate lr, a number of epochs epoc tolerance tol.
- The evaluate method gives the prediction for an X
- The score method gives the cost function for an X and y

```
class Model:
    def __init__(self, layers, loss):
        self.layers = layers
        self.loss = loss

    def fit(self, X, y, lr, epochs, batch_size, tol,print_loss = True):
        if self.loss.history:
            self.initialize()
        total_loss = np.inf
        j = 0
        while j <= epochs and total_loss >= tol:
            _loss = 0
            batches = len(y)//batch_size
            for i in range(batches):
                y_pred = self.forward_pass(X[i*batch_size:(i+1)*batch_size])
                loss_ = self.loss.forward_pass(y_pred,y[i*batch_size:(i+1)*batch_size
                loss_gradient = loss.backward_pass(y_pred, y[i*batch_size:(i+1)*batch
                self.backward_pass(loss_gradient)
                self.update_parameters(lr)
                _loss += loss_
            total_loss = _loss/batches
            loss.history.append(total_loss)
            self.update_gradient_history()
```

```
            if print_loss:
                print(f'epoch {j}: loss {total_loss}')
            j += 1

    def initialize(self):
        for layer in self.layers:
            if layer.trainable:
                layer.initialize()
        self.loss.initialize()

    def forward_pass(self, X):
        x = X.copy()
        for layer in self.layers:
            a = layer.forward_pass(x)
            x = a
        return x

    def backward_pass(self, upstream_gradient):
        for layer in reversed(self.layers):
            upstream_gradient = layer.backward_pass(upstream_gradient)
        return upstream_gradient

    def update_parameters(self,lr):
        for layer in self.layers:
            if layer.trainable:
                layer.update_parameters(lr)

    def update_gradient_history(self):
        for layer in self.layers:
            if layer.trainable:
                layer.gradient_history.append([np.sum(layer.gradient_history_w),np.su
                layer.gradient_history_w = []
                layer.gradient_history_b = []

    def evaluate(self, X):
        return self.forward_pass(X)

    def score(self, X, y):
        y_pred = self.forward_pass(X)
        loss = self.loss.forward_pass(y_pred,y)
        return loss
```

Layer

Each layer has its own initialization (in which we specify whether or not the layer is trainable) and has
for the backpropagation algorithm

```
class Layer:
    def __init__(self):
```

```
        pass

    def forward_pass(self, x):
        pass

    def backward_pass(self, upstream_gradient):
        pass
```

Local layer

Each local layer has the weights and bias for an input, and gives an output to be activated with any of trainable layer, that means it has a method update_parameters in which the gradient descend algorit (weights and bias) taking into account the local gradient stored

```
class Local(Layer):
    def __init__(self, input_size, output_size, bias = True):
        Xstdd = 2 / (input_size + output_size)
        self.weights = np.random.normal(loc = 0, scale = Xstdd, size = (input_size, o
        if bias:
            self.bias = np.zeros(output_size)
            self.has_bias = True
        else:
            self.has_bias = False
        self.local_gradient = {}
        self.x = None
        self.trainable = True
        self.gradient_history = []
        self.input_size = input_size
        self.output_size = output_size
        self.gradient_history_w = []
        self.gradient_history_b = []

    def initialize(self):
        Xstdd = 2 / (self.input_size + self.output_size)
        self.weights = np.random.normal(loc = 0, scale = Xstdd, size = (self.input_si
        if self.has_bias:
            self.bias = np.zeros(self.output_size)
        self.local_gradient = {}
        self.gradient_history = []
        self.gradient_history_w = []
        self.gradient_history_b = []

    # x: (batch_size, input_size)
    # w: (input_size, output_size)
    def forward_pass(self,_x):
        self.x = _x.copy()
        return _x @ self.weights + self.bias

    def backward_pass(self,upstream_gradient):
```

```
        dx = upstream_gradient @ self.weights.T
        dw = self.x.T @ upstream_gradient
        if self.has_bias:
            db = np.sum(upstream_gradient, axis = 0)
            self.local_gradient = {'dw': dw, 'db': db}
            self.gradient_history_b.append(np.sum(db))
        else:
            self.local_gradient = {'dw': dw}
        self.gradient_history_w.append(np.sum(dw))
        return dx

    def update_parameters(self,lr):
        self.weights = self.weights - lr*self.local_gradient['dw']
        if self.has_bias:
            self.bias = self.bias - lr*self.local_gradient['db']
```

Activation layers

Each activation layer has the forward and backward pass, an is not trainable. Currently available laye (Tanh). For a linear activation, do not set any activation.

```
class Sigmoid(Layer):
    def __init__(self):
        self.x = None
        self.trainable = False

    def forward_pass(self,_x):
        self.x = _x.copy()
        return 1 / (1 + np.exp(-_x))

    def backward_pass(self,upstream_gradient):
        s_prime = (1 / (1 + np.exp(-self.x)))*(1-(1 / (1 + np.exp(-self.x))))
        dx = upstream_gradient * s_prime
        return dx


class Relu(Layer):
    def __init__(self):
        self.x = None
        self.trainable = False

    def forward_pass(self,x):
        self.x = x
        return np.where(x > 0, x, 0)

    def backward_pass(self,upstream_gradient):
        r_prime = (self.x > 0).astype(np.float32)
        dx = upstream_gradient * r_prime
        return dx
```

```
class Tanh(Layer):
    def __init__(self):
        self.x = None
        self.trainable = False

    def forward_pass(self,x):
        self.x = x
        return np.tanh(x)

    def backward_pass(self,upstream_gradient):
        tan = np.tanh(self.x)
        dx = 1 - np.power(tan,2)
        return dx
```

Loss layer

It receives the output of the perceptron and calculates the loss, as well as the gradient passing troug history, so we can explore the cost in different stages of the training.

```
class Loss:
    def __init__(self):
        self.history = []

    def forward_pass(self, y_pred, y_true):
        pass

    def backward_pass(self):
        pass

    def initialize(self):
        self.history = []


class RegressionLoss(Loss):
    def forward_pass(self, y_pred, y_true):
        a = np.sum((y_pred - y_true)**2)
        return a / len(y_true)

    def backward_pass(self,y_pred,y_true):
        return (1/len(y_true))*(2 * (y_pred - y_true))
```

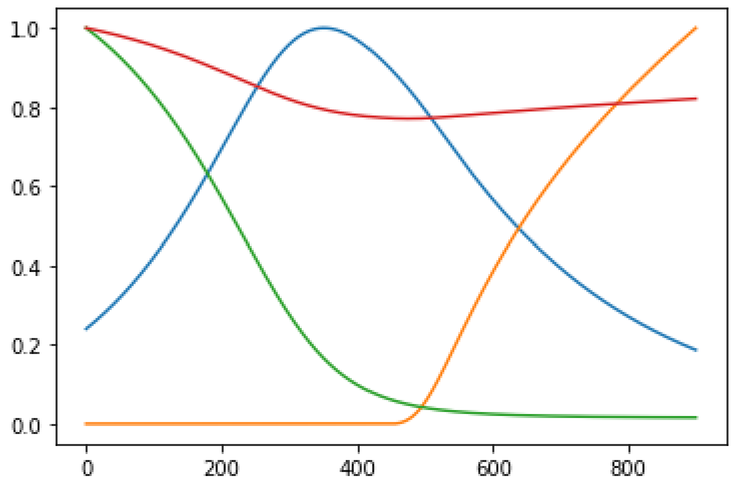
Preprocessing

Data loading and normalization

```
data = pd.read_csv("examen.csv", header = None)
```

```
X, y = data.loc[:,1:4].values, data.loc[:,0].values
X = X / np.max(X, axis = 0)
y = data[0] / max(data[0])
y = y.values

plt.plot(X)
plt.show()
```



Data splitting

We need three sets of data: training, validation and test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, r

M = X_train.shape[0] # Number of samples
m = X_train.shape[1] # Number of features or entries
n = 1 # number of outputs

l = 2 # Hidden layers
i = 5 # Number of neurons in each hidden layer
L = []
inp = m
activation_functions = [Sigmoid(), Sigmoid(), Sigmoid(), Sigmoid()]
for j in range(l+1):
    a = Local(inp,i)
    L.append(Local(inp,i))
    L.append(activation_functions[j])
    inp = i
L.append(Local(inp,1))
```

```
L[0].weights = np.array([[-0.1337,  0.0874, -0.1069, -0.2706],
        [ 0.1733,  0.3348,  0.2583,  0.1232],
        [-0.0607, -0.1557, -0.3172,  0.3525],
        [-0.2465,  0.3329, -0.0024,  0.2357],
        [ 0.4099,  0.1306, -0.1127, -0.0377]]).T


L[2].weights = np.array([[ 0.2459,  0.4003, -0.2752,  0.0781, -0.4327],
        [-0.2687, -0.1221, -0.0464, -0.2257, -0.2737],
        [ 0.2551, -0.1137,  0.0599, -0.1505,  0.2294],
        [-0.3911, -0.2979, -0.0358, -0.4079, -0.1854],
        [-0.1694, -0.2072, -0.3281, -0.3312,  0.3809]]).T


L[4].weights = np.array([[-0.3884, -0.3393, -0.0121, -0.2346,  0.2158],
        [ 0.2595, -0.1074,  0.4093,  0.3875, -0.2710],
        [ 0.2406, -0.1602, -0.3381, -0.4144,  0.2412],
        [ 0.0811, -0.1548,  0.0645, -0.2701, -0.0554],
        [ 0.1744,  0.3234,  0.1110,  0.2217, -0.0673]]).T


L[6].weights = np.array([[-0.2442,  0.1191,  0.3998, -0.4047,  0.4461]]).T


#L = [Local(4,10), Relu(), Local(10,15), Relu(), Local(15,1)]


loss = RegressionLoss()
model = Model(L,loss)


lr1 = 0.2
lr2 = 0.9
```

First learning rate

```
model.fit(X_train, y_train.reshape((M,1)), 0.01, 50, M, 1e-2)
```

```
epoch 0: loss 0.14081205216507664
epoch 1: loss 0.14078885698192367
epoch 2: loss 0.14076763914757418
epoch 3: loss 0.14074822991771344
epoch 4: loss 0.14073047496038685
epoch 5: loss 0.1407142331235354
epoch 6: loss 0.1406993753081165
epoch 7: loss 0.1406857834377407
epoch 8: loss 0.14067334951653587
epoch 9: loss 0.14066197476766595
epoch 10: loss 0.1406515688455836
epoch 11: loss 0.1406420491156909
epoch 12: loss 0.14063333999562852
epoch 13: loss 0.1406253723529094
epoch 14: loss 0.1406180829540683
epoch 15: loss 0.1406114139609128
epoch 16: loss 0.14060531246984193
epoch 17: loss 0.14059973009054302
epoch 18: loss 0.14059462256069663
epoch 19: loss 0.1405899493936059
epoch 20: loss 0.14058567355593396
epoch 21: loss 0.1405817611729722
epoch 22: loss 0.14057818125908483
epoch 23: loss 0.1405749054711766
epoch 24: loss 0.14057190788321366
epoch 25: loss 0.14056916477999923
epoch 26: loss 0.1405666544685566
epoch 27: loss 0.14056435710561532
epoch 28: loss 0.14056225453982393
epoch 29: loss 0.14056033016743127
epoch 30: loss 0.14055856880028522
epoch 31: loss 0.14055695654509695
epoch 32: loss 0.14055548069300847
epoch 33: loss 0.1405541296185831
epoch 34: loss 0.1405528926874146
epoch 35: loss 0.14055176017161908
epoch 36: loss 0.14055072317253597
```

Energy of the instant error

```
epoch 40: loss 0.14054737759828426
```

```
plt.plot(loss.history)
plt.show()
```


```
model.evaluate(X_train)
```

```
array([[0.40937767],
       [0.40926539],
       [0.40970973],
       [0.40988558],
       [0.40922878],
       [0.40967471],
       [0.40928887],
       [0.40998756],
       [0.41006465],
       [0.40931756],
       [0.4098471 ],
       [0.4092496 ],
       [0.40965062],
       [0.40985152],
       [0.40941179],
       [0.4093166 ],
       [0.40948955],
       [0.4093601 ],
       [0.41000794],
       [0.40972781],
       [0.40940567],
       [0.40973685],
       [0.40939024],
       [0.40967964],
       [0.40928443],
       [0.41001819],
       [0.40991623],
       [0.40996326],
       [0.40925451],
       [0.40968033],
       [0.40972043],
       [0.40921259],
       [0.40934537],
       [0.40979735],
       [0.41010957],
       [0.40969374],
       [0.40968374],
       [0.40946753],
       [0.40919171],
       [0.40976215],
       [0.40929336],
       [0.40936769],
       [0.40963314],
       [0.40959044],
       [0.40968426],
       [0.40921183],
       [0.40925866],
       [0.40922333],
       [0.40999612],
       [0.40982526],
       [0.40976565],
       [0.40929698],
       [0.40990028],
```

```
       [0.40965928],
       [0.40971308],
       [0.40962707],
       [0.40962551],
       [0.41009899],
       [0.40927222],
       [0.409792  ],
       [0.40989588],
       [0.40967981],
       [0.40951438],
       [0.40964436],
       [0.41005821],
       [0.40968947],
       [0.40998258],
       [0.40975536],
       [0.40990756],
       [0.40966823],
       [0.40969641],
       [0.40973042],
       [0.40967672],
       [0.40970658],
       [0.40968255],
       [0.40968982],
       [0.40932141],
       [0.40980553],
       [0.40978417],
       [0.40977907],
       [0.40982814],
       [0.4098678 ],
       [0.40951279],
       [0.40923748],
       [0.40933926],
       [0.40976802],
       [0.40930526],
       [0.40968069],
       [0.40970363],
       [0.41000445],
       [0.40923194],
       [0.40999248],
       [0.40948804],
       [0.40970719],
       [0.40933419],
       [0.40919908],
       [0.4098767 ],
       [0.41011357],
       [0.409796  ],
       [0.41009069],
       [0.40953556],
       [0.40956441],
       [0.40993608],
       [0.40929882],
       [0.40943076],
       [0.40958697],
       [0.40970538],
```

```
       [0.4094806 ],
       [0.40967888],
       [0.40920428],
       [0.41010189],
       [0.40971449],
       [0.40974371],
       [0.40920505],
       [0.40937208],
       [0.40930065],
       [0.40961589],
       [0.40977409],
       [0.40979068],
       [0.4095507 ],
       [0.41007521],
       [0.41011553],
       [0.4094661 ],
       [0.40943468],
       [0.41007691],
       [0.4100193 ],
       [0.40940811],
       [0.40988705],
       [0.40967648],
       [0.40987225],
       [0.41009452],
       [0.40924796],
       [0.40954393],
       [0.40966888],
       [0.40967948],
       [0.40968108],
       [0.40922957],
       [0.40931564],
       [0.40975208],
       [0.40973312],
       [0.40969093],
       [0.40953889],
       [0.41004061],
       [0.40968479],
       [0.40934127],
       [0.41004163],
       [0.41008755],
       [0.40999852],
       [0.40925287],
       [0.40929155],
       [0.40955582],
       [0.41010889],
       [0.40991479],
       [0.4100114 ],
       [0.40924389],
       [0.40994713],
       [0.410093  ],
       [0.40984563],
       [0.40971378],
       [0.40995795],
       [0.40945626],
```

```
       [0.40989   ],
       [0.40973974],
       [0.40976922],
       [0.40975101],
       [0.40968651],
       [0.4094987 ],
       [0.41002697],
       [0.40935163],
       [0.409684  ],
       [0.40921412],
       [0.40969933],
       [0.40977532],
       [0.40937543],
       [0.40926965],
       [0.40969417],
       [0.40971105],
       [0.40934744],
       [0.40923909],
       [0.40978546],
       [0.41005442],
       [0.409549  ],
       [0.40966377],
       [0.40944667],
       [0.40977043],
       [0.40983538],
       [0.40977286],
       [0.40969019],
       [0.40941927],
       [0.40928618],
       [0.4098604 ],
       [0.41007263],
       [0.40923511],
       [0.40947766],
       [0.40942433],
       [0.40989295],
       [0.40967861],
       [0.41008435],
       [0.40973497],
       [0.4093644 ],
       [0.41011024],
       [0.41003336],
       [0.4098153 ],
       [0.40938447],
       [0.41003959],
       [0.40971172],
       [0.4094704 ],
       [0.409761  ],
       [0.40987966],
       [0.40962076],
       [0.40941801],
       [0.40967624],
       [0.41001481],
       [0.40929065],
       [0.40998133],
```

```
       [0.40928709],
       [0.40968877],
       [0.40940326],
       [0.40950024],
       [0.40932825],
       [0.40922567],
       [0.41002371],
       [0.40925951],
       [0.40929426],
       [0.41001706],
       [0.40972281],
       [0.41008516],
       [0.4095696 ],
       [0.40973404],
       [0.40936333],
       [0.40968681],
       [0.40967751],
       [0.41005057],
       [0.41005537],
       [0.40927827],
       [0.40926033],
       [0.40945488],
       [0.40963892],
       [0.40938906],
       [0.41002151],
       [0.41007606],
       [0.40943206],
       [0.40967173],
       [0.40997368],
       [0.40957655],
       [0.4099724 ],
       [0.40985891],
       [0.40969332],
       [0.40929246],
       [0.41005727],
       [0.41001594],
       [0.40950804],
       [0.40960248],
       [0.40960925],
       [0.40969056],
       [0.40968301],
       [0.40942561],
       [0.40970845],
       [0.40968506],
       [0.40971814],
       [0.41007859],
       [0.40999126],
       [0.41008991],
       [0.4097378 ],
       [0.40947328],
       [0.40963165],
       [0.41006375],
       [0.4098224 ],
       [0.40972611],
```

```
[0.41009527],
[0.40928353],
[0.4098841 ],
[0.40923036],
[0.40970598],
[0.40933122],
[0.40939257],
[0.40969833],
[0.40993468],
[0.40985743],
[0.40920728],
[0.4097987 ],
[0.40920205],
[0.40989882],
[0.40930342],
[0.40937098],
[0.40970142],
[0.40938104],
[0.4095473 ],
[0.40972122],
[0.4095524 ],
[0.40924878],
[0.4092327 ],
[0.40930249],
[0.40966613],
[0.40992481],
[0.40969131],
[0.40933824],
[0.40985005],
[0.40938793],
[0.40933023],
[0.40977781],
[0.40965295],
[0.4093788 ],
[0.4096375 ],
[0.40953722],
[0.40935058],
[0.40964303],
[0.40994301],
[0.40932432],
[0.40946896],
[0.40973877],
[0.41004664],
[0.40938561],
[0.4092809 ],
[0.40975871],
[0.40930155],
[0.40924471],
[0.40968051],
[0.40920055],
[0.40922643],
[0.4092705 ],
[0.4099788 ],
[0.40975759]
```

```
       [0.40973739],
       [0.41003019],
       [0.40997753],
       [0.40968843],
       [0.41008913],
       [0.40990902],
       [0.40971038],
       [0.40981812],
       [0.40943731],
       [0.40967357],
       [0.40928531],
       [0.40978161],
       [0.4093961 ],
       [0.40978675],
       [0.40972955],
       [0.40944397],
       [0.41005632],
       [0.40999491],
       [0.40946185],
       [0.40963014],
       [0.40974473],
       [0.40944532],
       [0.40988262],
       [0.40985299],
       [0.40956269],
       [0.40920279],
       [0.40918806],
       [0.40981671],
       [0.40941303],
       [0.40949716],
       [0.40948356],
       [0.4095208 ],
       [0.40922488],
       [0.40968278],
       [0.40930995],
       [0.40927915],
       [0.40971739],
       [0.40937429],
       [0.40935797],
       [0.40968168],
       [0.40925785],
       [0.40983393],
       [0.40995257],
       [0.40939966],
       [0.4100735 ],
       [0.40979467],
       [0.4096881 ],
       [0.40964942],
       [0.40974575],
       [0.40930435],
       [0.40961258],
       [0.40968777],
       [0.4097124 ],
       [0.40938677],
       [0.40967506],
```

```
       [0.40967506],
       [0.41000561],
       [0.41010261],
       [0.41005346],
       [0.40928977],
       [0.40974679],
       [0.40927307],
       [0.4094941 ],
       [0.40931467],
       [0.40927135],
       [0.40950647],
       [0.40953224],
       [0.4093362 ],
       [0.41007776],
       [0.40986484],
       [0.40967694],
       [0.41008355],
       [0.40942306],
       [0.40970908],
       [0.40930713],
       [0.40958176],
       [0.41009825],
       [0.40942818],
       [0.4099711 ],
       [0.40936987],
       [0.40994577],
       [0.40936659],
       [0.40927395],
       [0.41009603],
       [0.40970421],
       [0.4098267 ],
       [0.4099485 ],
       [0.40990466],
       [0.40980969],
       [0.40968147],
       [0.40950179],
       [0.4096518 ],
       [0.40967314],
       [0.40992765],
       [0.40996589],
       [0.4100525 ],
       [0.40967903],
       [0.40919024],
       [0.40947184],
       [0.41002261],
       [0.41004465],
       [0.40933521],
       [0.40928796],
       [0.40961914],
       [0.40965728],
       [0.40923115],
       [0.40964031],
       [0.40957481],
       [0.40940689],
```

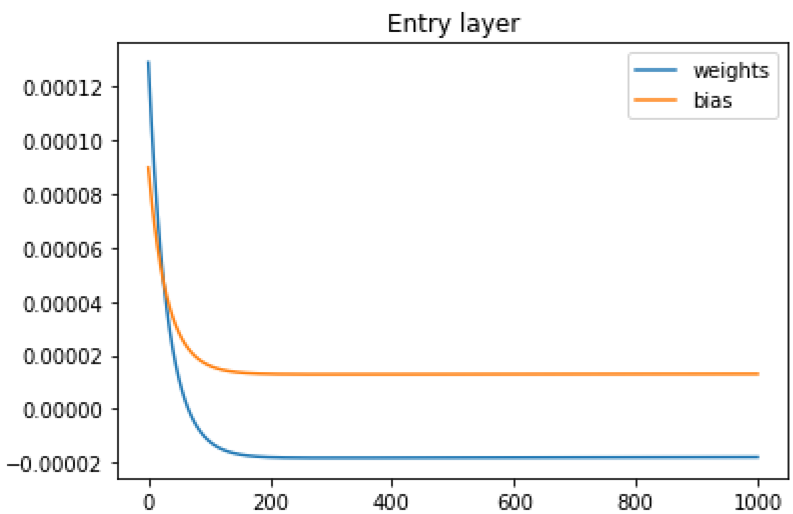
```
       [0.40982958],
       [0.40968233],
       [0.41010117],
       [0.40927739],
       [0.40925042],
       [0.40966537],
       [0.40925536],
       [0.40999971],
       [0.4098083 ],
       [0.40992906],
       [0.40981109],
       [0.40967822],
       [0.410061  ],
       [0.41003545],
       [0.41011617],
       [0.4095939 ],
       [0.40930619],
       [0.4093332 ],
       [0.40969504],
       [0.40967809],
       [0.40944802],
       [0.40977657],
       [0.40935372],
       [0.40952567],
       [0.40966293],
       [0.40934025],
       [0.40969984],
       [0.40929791],
       [0.40987522],
       [0.40945075],
       [0.409257  ],
       [0.40919982],
       [0.40932531],
       [0.40919097],
       [0.40959562],
       [0.40918733],
       [0.41008596],
       [0.41003753],
       [0.40957134],
       [0.40943996],
       [0.40986632],
       [0.41010403],
       [0.40983248],
       [0.4096921 ],
       [0.41010683],
       [0.40984416],
       [0.4097417 ],
       [0.40962394],
       [0.40922101],
       [0.40972695],
       [0.40992624],
       [0.40995392],
       [0.41008676],
       [0.40924228],
```

```
[0.40986187],
[0.40983103],
[0.4099191 ],
[0.40972201],
[0.40971665],
[0.40980415],
[0.4099937 ],
[0.40978034],
[0.40936117],
[0.40998508],
[0.40954056],
```

Local gradient evolution

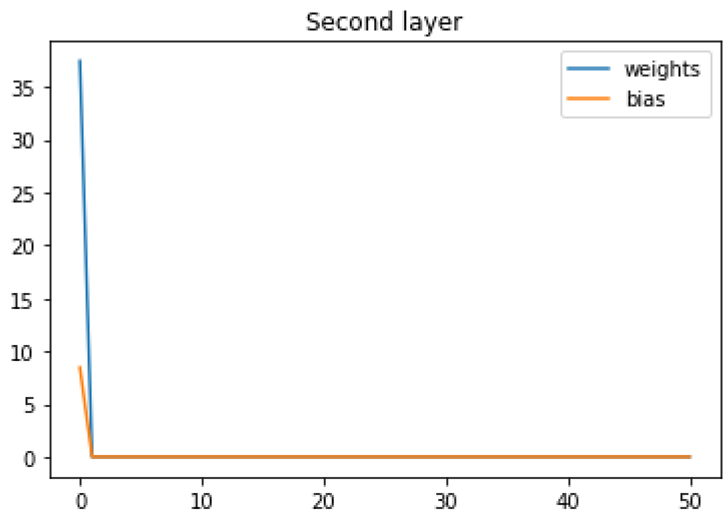
```
[0.40964168]
```

```
plt.plot(L[0].gradient_history)
plt.legend(['weights', 'bias'])
plt.title('Entry layer')
plt.show()
```

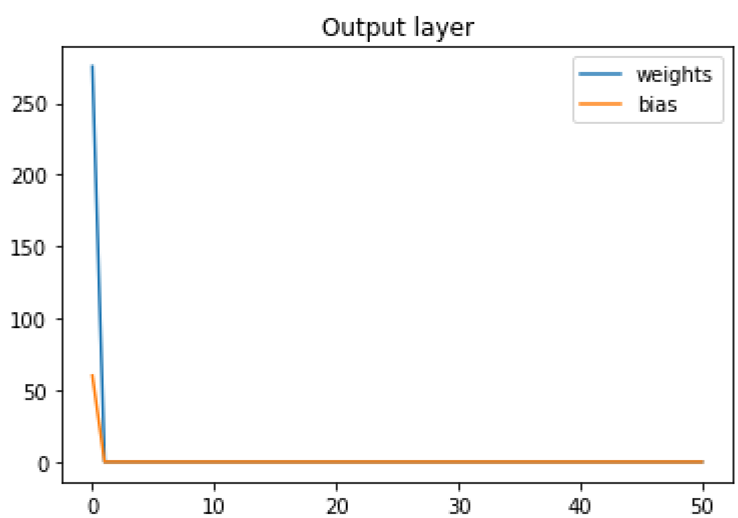



```
plt.plot(L[2].gradient_history)
plt.legend(['weights', 'bias'])
plt.title('First layer')
plt.show()
```


```
plt.plot(L[4].gradient_history)
plt.legend(['weights', 'bias'])
plt.title('Second layer')
plt.show()
```



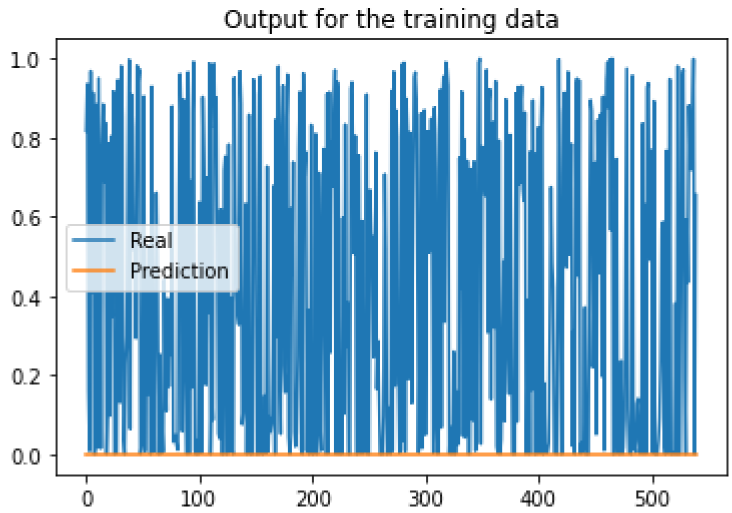

```
plt.plot(L[6].gradient_history)
plt.legend(['weights', 'bias'])
plt.title('Output layer')
plt.show()
```



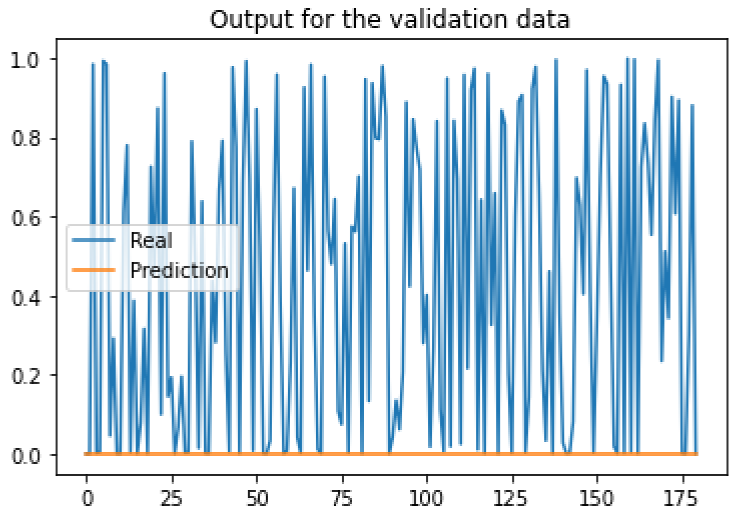
Predictions

```
plt.plot(y_train, label = 'Real')
plt.plot(model.evaluate(X_train), label = 'Prediction')
plt.title('Output for the training data')
```

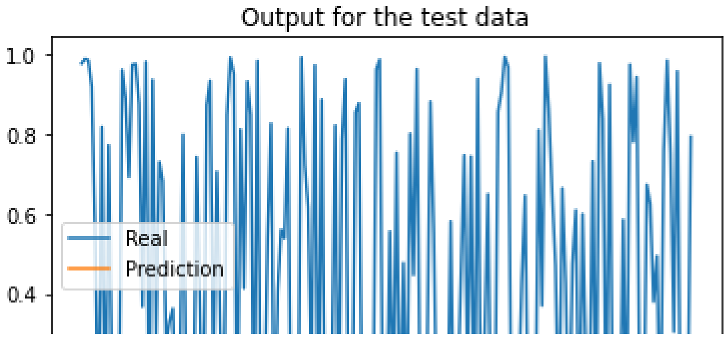
```
plt.legend()
plt.show()
```




```
plt.plot(y_val, label = 'Real')
plt.plot(model.evaluate(X_val), label = 'Prediction')
plt.title('Output for the validation data')
plt.legend()
plt.show()
```




```
plt.plot(y_test, label = 'Real')
plt.plot(model.evaluate(X_test), label = 'Prediction')
plt.title('Output for the test data')
plt.legend()
plt.show()
```



Cross validation

```
scores = []
cv = KFold(n_splits=10, shuffle = True, random_state = 42)
for train_index, test_index in cv.split(X_val):
    loss = RegressionLoss()
    model = Model(L,loss)
    X_train_v, X_test_v, y_train_v, y_test_v = X[train_index], X[test_index], y[trair
    model.fit(X_train_v, y_train_v.reshape((len(y_train_v),1)), lr1, 50, len(y_train_
    scores.append(model.score(X_test_v, y_test_v.reshape((len(y_test_v),1))))


print('Cross validation score:', np.mean(scores))
```

```
Cross validation score: 0.1999999941961233
```

Second learning rate

```
model.fit(X_train, y_train.reshape((M,1)), lr2, 50, M, 1e-2)
```

```
epoch 0: loss 54.57685166014407
epoch 1: loss 2.0
epoch 2: loss 2.0
epoch 3: loss 2.0
epoch 4: loss 2.0
epoch 5: loss 2.0
epoch 6: loss 2.0
epoch 7: loss 2.0
epoch 8: loss 2.0
epoch 9: loss 2.0
epoch 10: loss 2.0
epoch 11: loss 2.0
epoch 12: loss 2.0
epoch 13: loss 2.0
epoch 14: loss 2.0
epoch 15: loss 2.0
epoch 16: loss 2.0
epoch 17: loss 2.0
epoch 18: loss 2.0
epoch 19: loss 2.0
epoch 20: loss 2.0
epoch 21: loss 2.0
epoch 22: loss 2.0
epoch 23: loss 2.0
epoch 24: loss 2.0
epoch 25: loss 2.0
epoch 26: loss 2.0
epoch 27: loss 2.0
epoch 28: loss 2.0
epoch 29: loss 2.0
epoch 30: loss 2.0
epoch 31: loss 2.0
epoch 32: loss 2.0
```

Energy of the instant error

↳ *1 celda oculta*

Gradient history

↳ *7 celdas ocultas*

```
epoch 43: loss 2.0
```

Cross validation

↳ *3 celdas ocultas*

```
epoch 48: loss 2.0
epoch 49: loss 2.0
epoch 50: loss 2.0
```