Predicting Cancer Malignancy with a 2 layer neural network coded from scratch in Python.

This notebook holds the Python code connected to this 3 part article:

Part 1 | Part 2 | Part 3

With this code and the associated articles, you are going to:

- Create a neural network from scratch in Python. Train it using the gradient descent algorithm.
- Apply that basic network to The Wisconsin Cancer Data-set. Predict if a tumor is benign or malignant, based on 9 different features.
- Explore deeply how back-propagation and gradient descent work.
- Review the basics and explore advanced concepts.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.metrics import confusion_matrix
import itertools

np.set_printoptions(threshold=np.inf)

%matplotlib inline
%load_ext autoreload
%autoreload 2
```

```
def plotCf(a,b,t):
    cf =confusion_matrix(a,b)
    plt.imshow(cf,cmap=plt.cm.Blues,interpolation='nearest')
    plt.colorbar()
    plt.title(t)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    tick_marks = np.arange(len(set(a))) # length of classes
    class_labels = ['0','1']
    plt.xticks(tick_marks,class_labels)
    plt.yticks(tick_marks,class_labels)
    thresh = cf.max() / 2.
    for i,j in itertools.product(range(cf.shape[0]),range(cf.shape[1])):
        plt.text(j,i,format(cf[i,j],'d'),horizontalalignment='center',color='white' if cf[i,j] >thresh else 'black')
    plt.show();
```

The Dlnet 2 layer neural network class

A 2 layer neural network class with gradient descent in less than 100 lines of code

```
def Sigmoid(Z):
    return 1/(1+np.exp(-Z))

def Relu(Z):
    return np.maximum(0,Z)

def dRelu2(dZ, Z):
    dZ[Z <= 0] = 0
    return dZ

def dRelu(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x

def dSigmoid(Z):
    s = 1/(1+np.exp(-Z))
    dZ = s * (1-s)
    return dZ

class dlnet:
    def __init__(self, x, y):
        self.debug = 0;
        self.X=x
        self.Y=y
        self.Yh=np.zeros((1,self.Y.shape[1]))
        self.L=2
        self.dims = [9, 15, 1]
        self.param = {}
        self.ch = {}
        self.grad = {}
        self.loss = []
        self.lr=0.003
        self.sam = self.Y.shape[1]
        self.threshold=0.5

    def nInit(self):
        np.random.seed(1)
        self.param['W1'] = np.random.randn(self.dims[1], self.dims[0]) / np.sqrt(self.dims[0])
        self.param['b1'] = np.zeros((self.dims[1], 1))
        self.param['W2'] = np.random.randn(self.dims[2], self.dims[1]) / np.sqrt(self.dims[1])
        self.param['b2'] = np.zeros((self.dims[2], 1))
        return

    def forward(self):
        Z1 = self.param['W1'].dot(self.X) + self.param['b1']
        A1 = Relu(Z1)
        self.ch['Z1'],self.ch['A1']=Z1,A1

        Z2 = self.param['W2'].dot(A1) + self.param['b2']
        A2 = Sigmoid(Z2)
        self.ch['Z2'],self.ch['A2']=Z2,A2

        self.Yh=A2
        loss=self.nloss(A2)
        return self.Yh, loss

    def nloss(self,Yh):
        loss = (1./self.sam) * (-np.dot(self.Y,np.log(Yh).T) - np.dot(1-self.Y, np.log(1-Yh).T))
        return loss

    def backward(self):
        dLoss_Yh = - (np.divide(self.Y, self.Yh ) - np.divide(1 - self.Y, 1 - self.Yh))

        dLoss_Z2 = dLoss_Yh * dSigmoid(self.ch['Z2'])
        dLoss_A1 = np.dot(self.param["W2"].T,dLoss_Z2)
        dLoss_W2 = 1./self.ch['A1'].shape[1] * np.dot(dLoss_Z2,self.ch['A1'].T)
        dLoss_b2 = 1./self.ch['A1'].shape[1] * np.dot(dLoss_Z2, np.ones([dLoss_Z2.shape[1],1]))

        dLoss_Z1 = dLoss_A1 * dRelu(self.ch['Z1'])
        dLoss_A0 = np.dot(self.param["W1"].T,dLoss_Z1)
        dLoss_W1 = 1./self.X.shape[1] * np.dot(dLoss_Z1,self.X.T)
        dLoss_b1 = 1./self.X.shape[1] * np.dot(dLoss_Z1, np.ones([dLoss_Z1.shape[1],1]))

        self.param["W1"] = self.param["W1"] - self.lr * dLoss_W1
        self.param["b1"] = self.param["b1"] - self.lr * dLoss_b1
        self.param["W2"] = self.param["W2"] - self.lr * dLoss_W2
        self.param["b2"] = self.param["b2"] - self.lr * dLoss_b2

        return


    def pred(self,x, y):
        self.X=x
        self.Y=y
        comp = np.zeros((1,x.shape[1]))
        pred, loss= self.forward()

        for i in range(0, pred.shape[1]):
            if pred[0,i] > self.threshold: comp[0,i] = 1
            else: comp[0,i] = 0

        print("Acc: " + str(np.sum((comp == y)/x.shape[1])))

        return comp

    def gd(self,X, Y, iter = 3000):
        np.random.seed(1)

        self.nInit()

        for i in range(0, iter):
            Yh, loss=self.forward()
            self.backward()

            if i % 500 == 0:
                print ("Cost after iteration %i: %f" %(i, loss))
                self.loss.append(loss)
        plt.plot(np.squeeze(self.loss))
        plt.ylabel('Loss')
        plt.xlabel('Iter')
        plt.title("Lr =" + str(self.lr))
        plt.show()

        return
```

Prepare the data

Get the data from this link:
Wisconsin Cancer Dataset

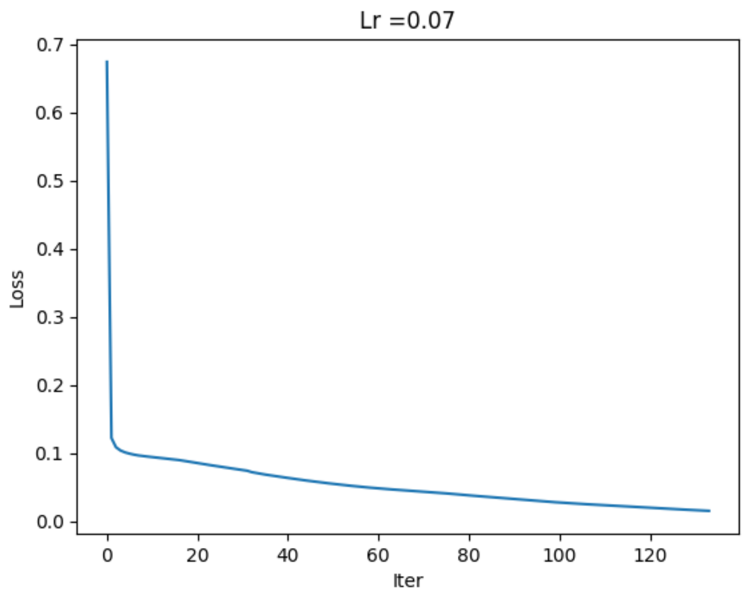
- Store the data in .csv format in your machine or online
- Read the data using Pandas read_csv function
- Then we proceed to clean and prepare the data, build our datasets and run gradient descent.

```
df = pd.read_csv('wisconsin-cancer-dataset.csv',header=None)
df = df[~df[6].isin(['?'])]
df = df.astype(float)
df.iloc[:,10].replace(2, 0,inplace=True)
df.iloc[:,10].replace(4, 1,inplace=True)

df.head(3)
scaled_df=df
names = df.columns[0:10]
scaler = MinMaxScaler()
scaled_df = scaler.fit_transform(df.iloc[:,0:10])
scaled_df = pd.DataFrame(scaled_df, columns=names)



x=scaled_df.iloc[0:500,1:10].values.transpose()
y=df.iloc[0:500,10:].values.transpose()

xval=scaled_df.iloc[501:683,1:10].values.transpose()
yval=df.iloc[501:683,10:].values.transpose()

print(df.shape, x.shape, y.shape, xval.shape, yval.shape)

nn = dlnet(x,y)
nn.lr=0.07
nn.dims = [9, 15, 1]
```

```
    (683, 11) (9, 500) (1, 500) (9, 182) (1, 182)
```

```
nn.gd(x, y, iter = 67000)
```

```
<ipython-input-3-4ecd3af0c5db>:107: DeprecationWarning: Conversion of an array with n
  print ("Cost after iteration %i: %f" %(i, loss))
Cost after iteration 0: 0.673967
Cost after iteration 500: 0.122093
Cost after iteration 1000: 0.108469
Cost after iteration 1500: 0.103673
Cost after iteration 2000: 0.100911
Cost after iteration 2500: 0.099047
Cost after iteration 3000: 0.097530
Cost after iteration 3500: 0.096368
Cost after iteration 4000: 0.095480
Cost after iteration 4500: 0.094744
Cost after iteration 5000: 0.094015
Cost after iteration 5500: 0.093277
Cost after iteration 6000: 0.092611
Cost after iteration 6500: 0.091953
Cost after iteration 7000: 0.091279
Cost after iteration 7500: 0.090472
Cost after iteration 8000: 0.089574
Cost after iteration 8500: 0.088575
Cost after iteration 9000: 0.087426
Cost after iteration 9500: 0.086303
Cost after iteration 10000: 0.085122
Cost after iteration 10500: 0.084010
Cost after iteration 11000: 0.083025
Cost after iteration 11500: 0.082001
Cost after iteration 12000: 0.080948
Cost after iteration 12500: 0.079923
Cost after iteration 13000: 0.078909
Cost after iteration 13500: 0.077922
Cost after iteration 14000: 0.076943
Cost after iteration 14500: 0.075973
Cost after iteration 15000: 0.074995
Cost after iteration 15500: 0.073944
Cost after iteration 16000: 0.071908
Cost after iteration 16500: 0.070665
Cost after iteration 17000: 0.069438
Cost after iteration 17500: 0.068322
Cost after iteration 18000: 0.067292
Cost after iteration 18500: 0.066307
Cost after iteration 19000: 0.065351
Cost after iteration 19500: 0.064394
Cost after iteration 20000: 0.063385
Cost after iteration 20500: 0.062418
Cost after iteration 21000: 0.061473
Cost after iteration 21500: 0.060553
Cost after iteration 22000: 0.059662
Cost after iteration 22500: 0.058792
Cost after iteration 23000: 0.057941
Cost after iteration 23500: 0.057108
Cost after iteration 24000: 0.056286
Cost after iteration 24500: 0.055492
Cost after iteration 25000: 0.054718
Cost after iteration 25500: 0.053962
Cost after iteration 26000: 0.053226
Cost after iteration 26500: 0.052496
Cost after iteration 27000: 0.051793
Cost after iteration 27500: 0.051108
Cost after iteration 28000: 0.050464
Cost after iteration 28500: 0.049839
Cost after iteration 29000: 0.049230
Cost after iteration 29500: 0.048652
Cost after iteration 30000: 0.048097
Cost after iteration 30500: 0.047555
Cost after iteration 31000: 0.047022
Cost after iteration 31500: 0.046492
Cost after iteration 32000: 0.045980
Cost after iteration 32500: 0.045480
Cost after iteration 33000: 0.044989
Cost after iteration 33500: 0.044509
Cost after iteration 34000: 0.044038
Cost after iteration 34500: 0.043574
Cost after iteration 35000: 0.043116
Cost after iteration 35500: 0.042659
Cost after iteration 36000: 0.042204
Cost after iteration 36500: 0.041756
Cost after iteration 37000: 0.041319
Cost after iteration 37500: 0.040703
Cost after iteration 38000: 0.040103
Cost after iteration 38500: 0.039508
Cost after iteration 39000: 0.038938
Cost after iteration 39500: 0.038385
Cost after iteration 40000: 0.037832
Cost after iteration 40500: 0.037278
Cost after iteration 41000: 0.036739
Cost after iteration 41500: 0.036198
Cost after iteration 42000: 0.035660
Cost after iteration 42500: 0.035130
Cost after iteration 43000: 0.034601
Cost after iteration 43500: 0.034066
Cost after iteration 44000: 0.033414
Cost after iteration 44500: 0.032821
Cost after iteration 45000: 0.032259
Cost after iteration 45500: 0.031707
Cost after iteration 46000: 0.031173
Cost after iteration 46500: 0.030656
Cost after iteration 47000: 0.030150
Cost after iteration 47500: 0.029661
Cost after iteration 48000: 0.029183
Cost after iteration 48500: 0.028709
Cost after iteration 49000: 0.028239
Cost after iteration 49500: 0.027777
Cost after iteration 50000: 0.027324
Cost after iteration 50500: 0.026883
Cost after iteration 51000: 0.026450
Cost after iteration 51500: 0.026030
Cost after iteration 52000: 0.025618
Cost after iteration 52500: 0.025215
Cost after iteration 53000: 0.024821
Cost after iteration 53500: 0.024434
Cost after iteration 54000: 0.024053
Cost after iteration 54500: 0.023679
Cost after iteration 55000: 0.023313
Cost after iteration 55500: 0.022952
Cost after iteration 56000: 0.022598
Cost after iteration 56500: 0.022250
Cost after iteration 57000: 0.021906
Cost after iteration 57500: 0.021568
Cost after iteration 58000: 0.021236
Cost after iteration 58500: 0.020776
Cost after iteration 59000: 0.020371
Cost after iteration 59500: 0.020000
Cost after iteration 60000: 0.019671
Cost after iteration 60500: 0.019352
Cost after iteration 61000: 0.018956
Cost after iteration 61500: 0.018534
Cost after iteration 62000: 0.018150
Cost after iteration 62500: 0.017782
Cost after iteration 63000: 0.017409
Cost after iteration 63500: 0.017076
Cost after iteration 64000: 0.016762
Cost after iteration 64500: 0.016443
Cost after iteration 65000: 0.016081
Cost after iteration 65500: 0.015735
Cost after iteration 66000: 0.015406
Cost after iteration 66500: 0.015090
```

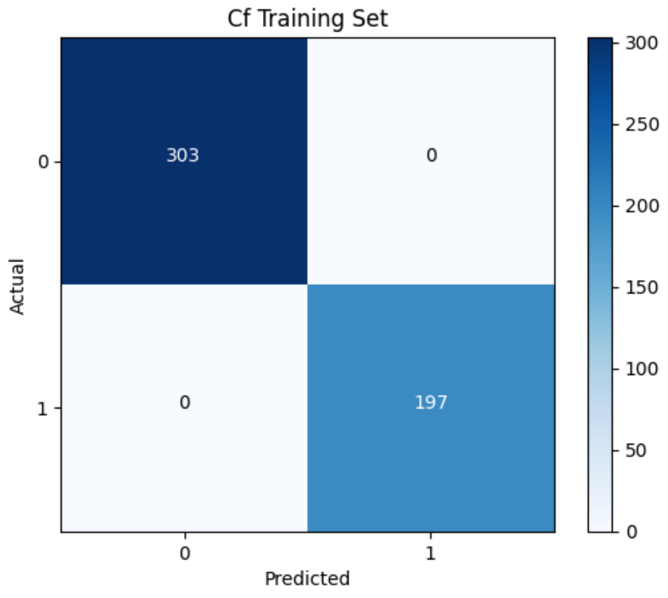


```
pred_train = nn.pred(x, y)
pred_test = nn.pred(xval, yval)
```

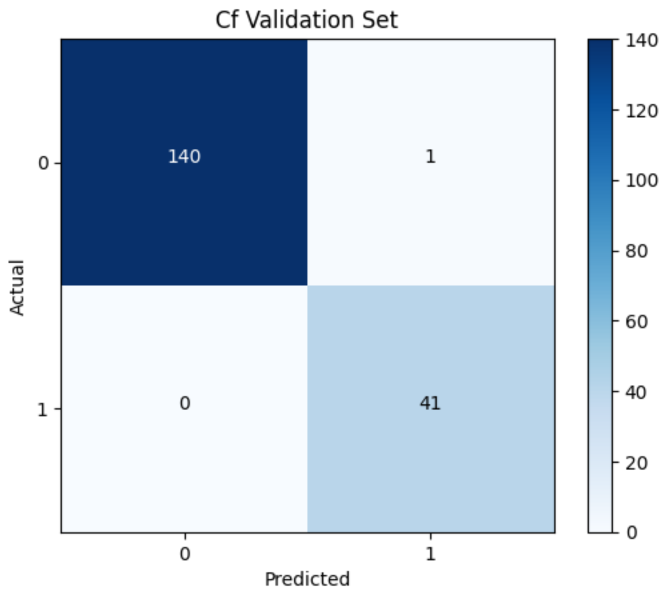
```
Acc: 1.0000000000000004
Acc: 0.9945054945054945
```

```
nn.threshold=0.5

nn.X,nn.Y=x, y
target=np.around(np.squeeze(y), decimals=0).astype(int)
predicted=np.around(np.squeeze(nn.pred(x,y)), decimals=0).astype(int)
plotCf(target,predicted,'Cf Training Set')

nn.X,nn.Y=xval, yval
target=np.around(np.squeeze(yval), decimals=0).astype(int)
predicted=np.around(np.squeeze(nn.pred(xval,yval)), decimals=0).astype(int)
plotCf(target,predicted,'Cf Validation Set')
```

```
    Acc: 1.0000000000000004
```

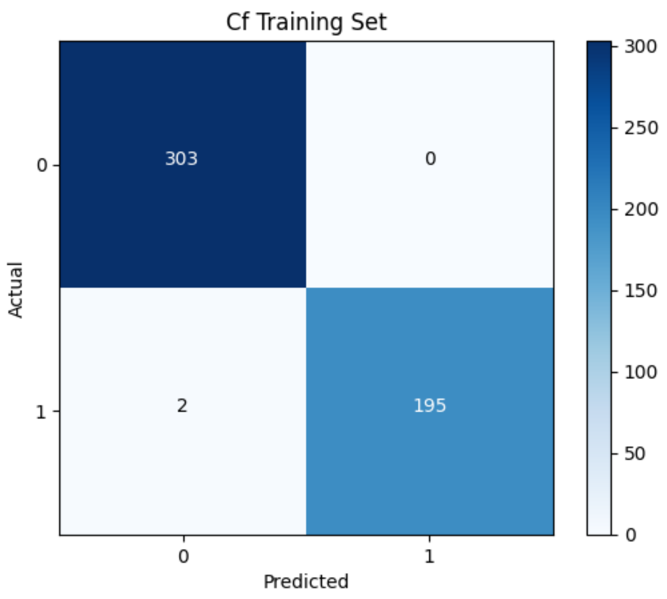


```
    Acc: 0.9945054945054945
```

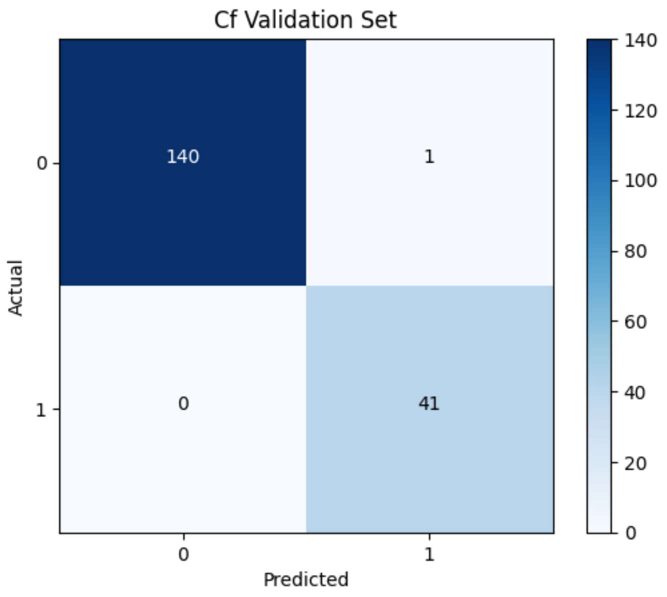


```
nn.threshold=0.7

nn.X,nn.Y=x, y
target=np.around(np.squeeze(y), decimals=0).astype(int)
predicted=np.around(np.squeeze(nn.pred(x,y)), decimals=0).astype(int)
plotCf(target,predicted,'Cf Training Set')

nn.X,nn.Y=xval, yval
target=np.around(np.squeeze(yval), decimals=0).astype(int)
predicted=np.around(np.squeeze(nn.pred(xval,yval)), decimals=0).astype(int)
plotCf(target,predicted,'Cf Validation Set')
```

```
Acc: 0.9960000000000003
```

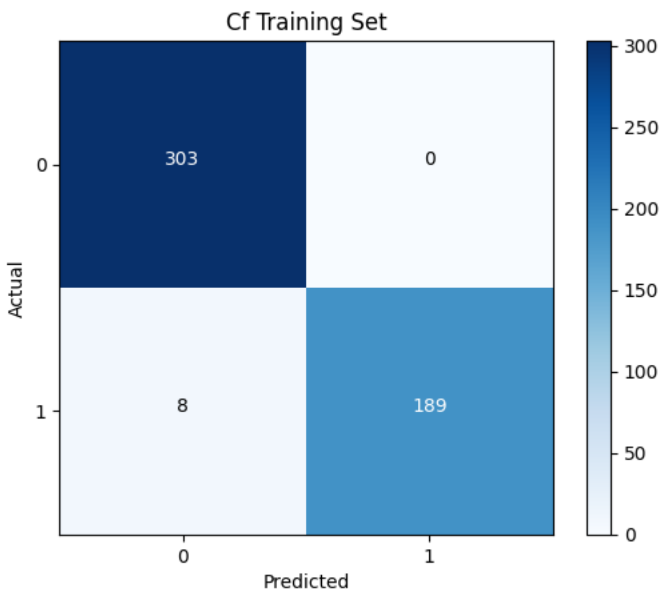


```
Acc: 0.9945054945054945
```



```
nn.threshold=0.9

nn.X,nn.Y=x, y
target=np.around(np.squeeze(y), decimals=0).astype(int)
predicted=np.around(np.squeeze(nn.pred(x,y)), decimals=0).astype(int)
plotCf(target,predicted,'Cf Training Set')

nn.X,nn.Y=xval, yval
target=np.around(np.squeeze(yval), decimals=0).astype(int)
predicted=np.around(np.squeeze(nn.pred(xval,yval)), decimals=0).astype(int)
plotCf(target,predicted,'Cf Validation Set')
```

```
Acc: 0.9840000000000003
```



```
Acc: 0.9945054945054945
```