PyTorch Neural Network Classification

# What is a classification problem?

A [classification problem](https://en.wikipedia.org/wiki/Statistical_classification) involves predicting whether something is one thing or another.

|  |  |  |
| --- | --- | --- |
| **Problem type** | **What is it?** | **Example** |
| **Binary classification** | Target can be one of two options, e.g. yes or no | Predict whether or not someone has heart disease based on their health parameters. |
| **Multi-class classification** | Target can be one of more than two options | Decide whether a photo of is of food, a person or a dog. |
| **Multi-label classification** | Target can be assigned more than one option | Predict what categories should be assigned to a Wikipedia article (e.g. mathematics, science & philosohpy). |

# What we are going to cover

|  |  |
| --- | --- |
| **Topic** | **Contents** |
| **0. Architecture of a classification neural network** | Neural networks can come in almost any shape or size, but they typically follow a similar floor plan. |
| **1. Getting binary classification data ready** | Data can be almost anything but to get started we're going to create a simple binary classification dataset. |
| **2. Building a PyTorch classification model** | Here we'll create a model to learn patterns in the data, we'll also choose a **loss function**, **optimizer** and build a **training loop** specific to classification. |
| **3. Fitting the model to data (training)** | We've got data and a model, now let's let the model (try to) find patterns in the (**training**) data. |
| **4. Making predictions and evaluating a model (inference)** | Our model's found patterns in the data, let's compare its findings to the actual (**testing**) data. |
| **5. Improving a model (from a model perspective)** | We've trained an evaluated a model but it's not working, let's try a few things to improve it. |
| **6. Non-linearity** | So far our model has only had the ability to model straight lines, what about non-linear (non-straight) lines? |
| **7. Replicating non-linear functions** | We used **non-linear functions** to help model non-linear data, but what do these look like? |
| **8. Putting it all together with multi-class classification** | Let's put everything we've done so far for binary classification together with a multi-class classification problem. |

# Technologies / libraries used for this session:

1. pandas -> for transforming raw data into data frames
2. scikit learn( sklearn ) 🡪 to get some sample public data and also for data visualization , also for splitting data into training and testing data sets
3. matplotlib 🡪 for plotting data graphically and data scattering
4. torch 🡪 tensors creation and neural network classes for creating layered neural networks

# 0. Architecture of a classification neural network

|  |  |  |
| --- | --- | --- |
| **Hyperparameter** | **Binary Classification** | **Multiclass classification** |
| **Input layer shape** (in\_features) | Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction) | Same as binary classification |
| **Hidden layer(s)** | Problem specific, minimum = 1, maximum = unlimited | Same as binary classification |
| **Neurons per hidden layer** | Problem specific, generally 10 to 512 | Same as binary classification |
| **Output layer shape** (out\_features) | 1 (one class or the other) | 1 per class (e.g. depends on how many output labels we are expecting . ex : for any photo if we are looking for RGB values then output layer will have 3 features) |
| **Hidden layer activation** | Usually ReLU (rectified linear unit) but can be many others | Same as binary classification |
| **Output activation** | Sigmoid (torch.sigmoid in PyTorch) | Softmax (torch.softmax in PyTorch) |
| **Loss function** | Binary crossentropy (torch.nn.BCELoss in PyTorch) | [Cross entropy (torch.nn.CrossEntropyLoss in PyTorch)](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fpytorch.org%2Fdocs%2Fstable%2Fgenerated%2Ftorch.nn.CrossEntropyLoss.html) |
| **Optimizer** | SGD (stochastic gradient descent), Adam (see torch.optim for more options) | Same as binary classification |

# 1.a. Make some classification data

We are using scikit learn here to get some data.

we are going to use make\_circles() method from scikit learn to generate 2 circles with different coloured dots.

from sklearn.datasets import make\_circles

# Make 1000 samples

n\_samples = 1000

# Create circles

X, y = make\_circles(n\_samples,

                    noise=0.03, # some noise for randomness

                    random\_state=42) # keep random state

X here is features and y are labels

Creating a data frame using pandas:

X values are getting partitioned into X1 and X2 below. For each x1,x2 values there is a Y value defined ( either 0,1)

# Make DataFrame of circle data

import pandas as pd

circles = pd.DataFrame({"X1": X[:, 0],

    "X2": X[:, 1],

    "label": y

})

circles.head(10)

Using matplot lib to plot the data set.

# Visualize with a plot

import matplotlib.pyplot as plt

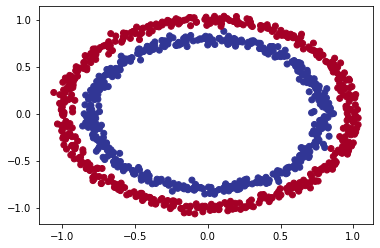
plt.scatter(x=X[:, 0],

            y=X[:, 1],

            c=y,

            cmap=plt.cm.RdYlBu);

1. less noise , more accurate and dots are more closer,
2. more noise , less accurate and dots are scattered around



# 1.b Input and output Shapes

# First see what the shape of your data set ( its important to understand the shapes and dimension of your data set since we will be dealing with tensor operations)

# Check the shapes of our features and labels

X.shape, y.shape

# View the first example of features and labels

X\_sample = X[0]

y\_sample = y[0]

print(f"Values for one sample of X: {X\_sample} and the same for y: {y\_sample}")

print(f"Shapes for one sample of X: {X\_sample.shape} and the same for y: {y\_sample.shape}")

# Here the X shape will be (1000,2) and y shape will be (1000,)

# ( 1000,2 ) means 100 data sets for X but its in pairs like [(x1,x2),(x3,x4)…..) and for y its in scaler( single number , 1 dimension) like [y1,y2,y3…..]

# 1000 🡪 count of data

# 2 🡪 numbers of features for the data( and it can vary and increase based on your data set)

# Converting data into tensors

# Turn data into tensors

# Otherwise this causes issues with computations later on

import torch

X = torch.from\_numpy(X).type(torch.float)

y = torch.from\_numpy(y).type(torch.float)

# View the first five samples

X[:5], y[:5]

# Splitting data into training and test sets

# Split data into train and test sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,

                                                    y,

                                                    test\_size=0.2, # 20% test, 80% train

                                                    random\_state=42) # make the random split reproducible

len(X\_train), len(X\_test), len(y\_train), len(y\_test)

# 2.Building a model

**Using GPU for neural networks**

Select your device first ( ie. CPU or GPU)

If you have a GPU, you can use that for running these models once created

The same will work on CPU (since the code is agnostic to which device its running on)

**Steps for model setup**

1. Subclasses nn.Module
2. Creates 2 nn.Linear layers in the constructor capable of handling the input and output shapes of X and y.
   1. Ie. Output of x , will be input of y
   2. In features 🡪 params going in the layer( since x has 2 params here , so keeping it 2)
   3. Out features 🡪 params going out from layer (This layer turns the input data from having 2 features to 5 features.)
      1. This is arbitrary number , also denotes no. of neurons per layer
      2. What’s the benefit of having out features ??
         1. This allows the model to learn patterns from 5 numbers rather than just 2 numbers, *potentially* leading to better outputs.
3. Defines a forward() method containing the forward pass computation of the model.

# 1. Construct a model class that subclasses nn.Module

class CircleModelV0(nn.Module):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # 2. Create 2 nn.Linear layers capable of handling X and y input and output shapes

        self.layer\_1 = nn.Linear(in\_features=2, out\_features=5) # takes in 2 features (X), produces 5 features

        self.layer\_2 = nn.Linear(in\_features=5, out\_features=1) # takes in 5 features, produces 1 feature (y)

    # 3. Define a forward method containing the forward pass computation

    def forward(self, x):

        # Return the output of layer\_2, a single feature, the same shape as y

        return self.layer\_2(self.layer\_1(x)) # computation goes through layer\_1 first then the output of layer\_1 goes through layer\_2

# 4. Create an instance of the model

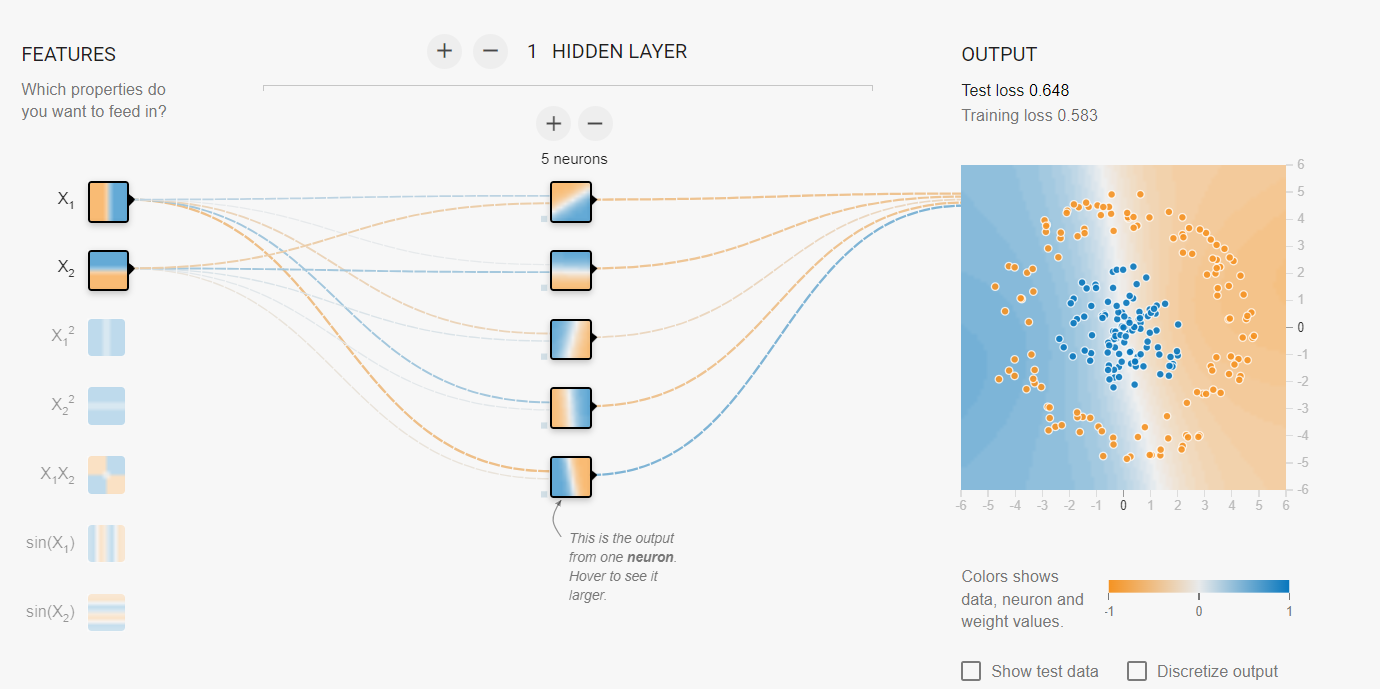
model\_0 = CircleModelV0()

**Tensor flow play ground ( for visualization)**

[playground.tensorflow.org](https://r.search.yahoo.com/_ylt=AwrKBzcu0N9kY5sULXG7HAx.;_ylu=Y29sbwNzZzMEcG9zAzEEdnRpZAMEc2VjA3Ny/RV=2/RE=1692418222/RO=10/RU=https%3a%2f%2fplayground.tensorflow.org%2f/RK=2/RS=RiIx9122z3SOJvNxVJ.bnTAOYQQ-)

2 input layers

1 hidden layer ( 5 neorons in hidden layer)



## nn.Sequential

The same model above can be constructed using nn.sequential module. This is class where output for one layer is given to another layer ( so data passes sequentially from one layer to another). This is similar to the custom class we made above (torch has an inbuilt class )

# Replicate CircleModelV0 with nn.Sequential

model\_0 = nn.Sequential(

    nn.Linear(in\_features=2, out\_features=5),

    nn.Linear(in\_features=5, out\_features=1)

)

model\_0

## #Make prediction with model

untrained\_preds = model\_0(X\_test)

print(f"Length of predictions: {len(untrained\_preds)}, Shape: {untrained\_preds.shape}")

print(f"Length of test samples: {len(y\_test)}, Shape: {y\_test.shape}")

print(f"\nFirst 10 predictions:\n{untrained\_preds[:10]}")

print(f"\nFirst 10 test labels:\n{y\_test[:10]}")

# 2.a Setup loss function and optimizer

Now based on the type of problem we define the loss function.

For instance, for regression , we chose MAE ( Mean Absolute error) and MSE ( Mean Squared Error).

For classification ( binary classification ) , we need binary cross entropy as the loss function.

<https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>